



POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Instytut Informatyki

Praca dyplomowa inżynierska

AUTOMATYCZNE WYKRYWANIE FAŁSZYWYCH TREŚCI ZA POMOCĄ ALGORYTMÓW UCZENIA MASZYNOWEGO

Igor Santarek, 154059

Mateusz Sztefek, 154026

Promotor

dr. inż. Dawid Wiśniewski

POZNAŃ 2025

Streszczenie

Niniejsza praca dotyczy klasyfikowania prawdziwości treści przy pomocy metod uczenia maszynowego i zbadania, czy wykorzystanie automatycznie wygenerowanych treści poprawi efektywność klasyfikacji. Do kodowania treści wykorzystano model RoBERTa [22]. Wygenerowaną treścią jest automatycznie zsyntetyzowany artykuł na podstawie analizowanej treści. W ramach tej pracy powstała również aplikacja demo, która służy do testowania naszego rozwiązania.

Abstract

This thesis concerns classifying the truthfulness of content using machine learning methods and examining whether the use of automatically generated content improves classification effectiveness. The RoBERTa model [22] was used to encode the content. The generated content is an automatically synthesized article based on the analyzed content. As part of this work, a demo application was also created, which serves to test our solution.

Spis treści

1	Wstęp	1
1.1	Motywacje i ważność problemu	1
1.2	Cel projektu	1
1.3	Zakres projektu	1
2	Podstawy teoretyczne	3
2.1	Znaczenie terminów w NLP	3
2.2	Zadania NLP	3
2.3	Struktury danych	4
2.4	Uczenie maszynowe	4
2.5	Sieci neuronowe	5
2.5.1	Optymalizator	5
2.5.2	Funkcje aktywacji	8
2.5.3	Dropout	10
2.5.4	Softmax	10
2.6	Przekleństwo wymiarowości i błogosławieństwo nierównomierności	10
2.7	Reprezentacja tekstu	11
2.8	Architektura Transformer	12
2.8.1	Model BERT	13
2.8.2	Model RoBERTa	15
2.9	Miary	15
2.9.1	Macierz pomyłek	16
2.9.2	Precyzyja i pełność	16

2.9.3	F-score	17
2.9.4	Dokładność	17
2.9.5	Rodzaje redukcji metryk	17
2.9.6	Entropia	18
2.10	Trudności związane z klasyfikacją prawdziwości treści	19
2.10.1	Złożoność formatów wejściowych	19
2.10.2	Istotność kontekstu informacji	20
2.10.3	Rodzaje fałszywych informacji	20
2.10.4	Halucynacja LLM	20
2.11	Technologie	20
3	Wybór zbioru danych	22
4	Eksploracyjna analiza danych	25
4.1	Opis zbioru danych	25
4.2	Generacja dodatkowych metadanych	27
4.3	Oczyszczenie i analiza zbioru danych	28
4.3.1	Oczyszczenie zbioru	28
4.3.2	Analiza zbioru	29
4.4	Normalizacja zbioru danych	39
5	Rozwiązańe	40
5.1	Proces treningu	40
5.2	Sposób trenowania	41
5.2.1	Zbieranie metryk	42
5.2.2	Mechanizm wczesnego zatrzymania	42
5.2.3	Mechanizm zapisu kroków pośrednich	42
5.2.4	Optymalizator	42
5.2.5	Funkcja straty	42
5.3	Bazowa implementacja klasyfikatora RoBERTa	43
5.3.1	Architektura	43
5.3.2	Trenowanie	44
5.4	Zbadanie możliwych wariantów wejścia	45
5.4.1	Architektura LiarPlusMultipleRoBERTasClassifier	46
5.4.2	Architektura LiarPlusSingleRoBERTaClassifier	48
5.4.3	Architektura LiarPlusSingleFinetunedRoBERTaClassifier	49
5.4.4	Architektura LiarPlusSingleRoBERTaDropoutClassifier	50
5.4.5	Porównanie i wybór najlepszej architektury	51
6	Rozszerzenie modelu o generację treści	56
6.1	Wybór modelu LLM do generowania treści	56
6.2	Proces generowania treści	56
6.3	Integracja generowanej treści z klasyfikatorem	57
7	Eksperymenty i wyniki	58
7.1	Architektura rozwiązań i ich motywacje	58
7.2	Porównanie klasyfikatorów	58
7.3	Analiza wpływu generowanych treści na dokładność klasyfikacji	60

7.3.1	Metodologia analizy	60
7.3.2	Ogólna analiza zbiorów	64
7.3.3	Szczegółowa analiza przykładów	66
7.3.4	Najbardziej różniące się przykłady ze zbioru SM_correct_SMA_incorrect . .	67
7.3.5	Najbardziej różniące się przykłady ze zbioru SM_incorrect_SMA_correct .	70
7.3.6	Podsumowanie szczegółowej analizy	74
8	Implementacja programu demonstracyjnego	76
8.1	Cel programu demonstracyjnego	76
8.2	Architektura systemu	76
8.3	Frontend i backend	76
8.4	Integracja z modelem klasyfikatora	78
9	Podsumowanie i wnioski	79
Literatura		81
Opis zawartości CD		86
Spis rysunków		86
Spis tabel		89

Rozdział 1

Wstęp

1.1 Motywacje i ważność problemu

W dobie powszechnego dostępu do internetu weryfikacja wciąż przybywających treści pod względem prawdziwości staje się coraz trudniejsza. Każdego dnia przybywa coraz więcej nowych użytkowników sieci, a w szczególności mediów społecznościowych, które zaczynają odgrywać kluczową rolę w kształtowaniu dyskursu publicznego [71]. Różni aktorzy zauważają ten trend i próbują poprzez wysublimowane mechanizmy psychologiczne wpływać na opinie publiczną, tak aby osiągać swoje cele na różnych polach. Wiele osób nieświadomie powiela dezinformacje. Aby temu zapobiegać istnieją różne organizacje fact-checkingowe, które weryfikują prawdziwość informacji. W Polsce takim zadaniem zajmuje się np. serwis demagog.org.pl.

W lutym 2022r Demagog [16] informował o zjawisku rozprzestrzeniania się w mediach społeczeństwowych informacji o rzekomych brakach paliwa na stacjach w wyniku inwazji Rosji na Ukrainę, przez co obywatele zaczęli w panice masowo wykupywać paliwo. W tym czasie niektórzy przedsiębiorcy, prowadzący stacje benzynowe, wykorzystali zaistniałą sytuację w celu manipulacji cenami marży. Wykorzystanie technik manipulacji i generowania fałszywych informacji jest poważnym zagrożeniem dla poprawnego działania demokracji.

Według autorów zbioru POLygraph: Polish Fake News Dataset [21], istnienie systemu automatycznej weryfikacji prawdziwości informacji mogłoby posłużyć jako wsparcie dla wielu instytucji, np. jednostkom sektora publicznego, takim jak Ministerstwo Spraw Wewnętrznych i Administracji, Ministerstwo Obrony, policji, i ABW w celu bezpieczeństwa publicznego; a także sektorowi prywatnemu, np. wydawcom, instytucjom finansowym takim jak Warszawska Giełda Papierów Wartościowych, Komisja Nadzoru Finansowego (w celu monitorowania potencjalnych manipulacji wpływających na wartość firmy albo stan makroekonomiczny państwa).

1.2 Cel projektu

Celem projektu jest implementacja systemu przetwarzania języka naturalnego, zdolnego do klasifikacji poziomu prawdziwości treści na podstawie cech semantycznych tekstu (np. na podstawie samej treści twierdzenia), a także zbadanie tego, czy rozszerzenie analizowanego przykładu – o treści wygenerowane przez duży model językowy – poprawi efektywność klasyfikacji.

1.3 Zakres projektu

Pracę realizujemy w zespole dwuosobowym. Dokładny zakres wykonanych zadań jest opisany w Tabeli 1.1.

TABELA 1.1: Spis wykonanych zadań w poszczególnych rozdziałach pracy

Rozdział	Igor Santarek	Mateusz Sztefek
1	Cały rozdział.	Poprawki do sekcji 1.3.
2	Wszystkie sekcje, poza 2.5.4. W 2.11 opisano Mlflow i LanguageTool.	Sekcje 2.5.4 i 2.11. W 2.11 opisano CUDA, Jupyter Notebook, DVC, Pandas, SQLite, Paramiko, TorchMetrics, scikit-learn i Gradio.
3	Sprawdzenie i opisanie zbiorów: wszystkie wymienione w rozdziale.	Sprawdzenie zbiorów: FEVER, NELAGT 2020, COVID-19 Fake News Dataset.
4	<p>Skrypty rozszerzające zbiór danych o informację o:</p> <ul style="list-style-type: none"> • sentymencie zawartym w treści • zliczaniu błędów gramatycznych w treści • byciu pytaniem lub nie • liczbie kropek na 100 znaków w treści • liczbie słów w treści i w uzasadnieniu przypisania do klasy <p>Analiza (plik <code>Basic analysis.ipynb</code>):</p> <ul style="list-style-type: none"> • zbalansowania zbioru • statystyk zbioru • słownika poszczególnych kolumn • odnośników WWW • liczby wypowiedzi na stan, autora, partię, pracę, temat • typów zdań • sentymentu zdań • średniej liczby błędów gramatycznych <p>Wykrycie i poprawa błędów w strukturze plików CSV zbioru danych. Wykrycie niepoprawnie przypisanych uzasadnień do przykładów w zbiorze danych. Normalizacja zbioru danych.</p>	<p>Skrypty rozszerzające zbiór danych o informację o:</p> <ul style="list-style-type: none"> • stronniczości politycznej wynikającej z treści • ofensywności treści • belkotliwości treści • nacechowania emocjonalnego w treści • poziomie wulgarności tekstu <p>Analiza (plik <code>EDA for new columns.ipynb</code>):</p> <ul style="list-style-type: none"> • ofensywności treści • belkotliwości treści • nacechowania emocjonalnego treści • wulgarności tekstu • stronniczości politycznej tekstu <p>Poprawienie kolumny <code>curse</code> w ramach normalizacji zbioru danych.</p>
5	Wszystko w tym rozdziale.	nic.
6	nic.	Wszystko w tym rozdziale.
7	Rozszerzenie inicjalnych modeli o wykorzystanie wektora one-hot, poprawa niektórych błędów, przeprowadzenie eksperymentów jeszcze raz, cała sekcja 7.3, włącznie z całą wykonaną pracą dotyczącą analizy w tym rozdziale.	Inicjalna wersja modeli, przeprowadzenie eksperymentów, całe sekcje 7.1 i 7.2.
8	nic.	Wszystko w tym rozdziale.

Rozdział 2

Podstawy teoretyczne

2.1 Znaczenie terminów w NLP

Przetwarzanie języka naturalnego (*Natural Language Processing* - NLP) to dziedzina sztucznej inteligencji, której celem jest całościowe zrozumienie tekstu naturalnego w kontekście obliczeniowym [70]. Poniżej są wypisane definicje najbardziej podstawowych pojęć w NLP [43]:

Tekst naturalny to tekst zapisany w języku naturalnym, czyli takim, którym posługują się ludzie.

Token to najmniejsza jednostka danych w NLP reprezentująca: słowo; część słowa; znak, z którego składa się słowo lub połączenia kilku słów (tzw. n-gramy). Mówiąc o tokenie, zwykle ma się na myśli wektor liczb (rzadki lub gęsty), ale również ciąg znaków, który reprezentuje ten wektor.

Słownik to zbiór reprezentujący wszystkie możliwe i jednocześnie rozważane tokeny. W zadaniu NLP definiuje się pewien słownik V , taki że token $\vec{t}_i \in V$. Model NLP może wykorzystywać inny słownik niż zbiór danych, na którym jest trenowany, np. zbiór danych zawiera N unikalnych słów (tokenów), a tokenizator modelu, który implementuje słownik o M unikalnych tokenach, może kodować te słowa na części słów. Dzięki temu M może być mniejsze niż N , a mimo tego wciąż być w stanie reprezentować wszystkie możliwe tokeny ze słownika zbioru danych.

Dokument to ciąg tokenów reprezentujący pewien tekst, np. artykuł, wiadomość lub fragment książki.

Korpus to zbiór dokumentów, który jest wykorzystywany do celów analizy języka naturalnego lub trenowania modeli NLP.

Tokenizator to kod komputerowy, który zamienia tekst naturalny na ciąg tokenów (dokonuje tzw. **tokenizacji**). Może oddzielać słowa w miejscach, gdzie są spacje, ale może też oddzielać je w innych miejscach, np. dzieląc słowa na części, a także może intencjonalnie pozostawiać połączenia słów przy pomocy spacji, np. „New York” może zostać zakodowane jako pojedynczy token.

Stop-words to bardzo często pojawiające się słowa w korpusie, np. „the”, „a”, itp...

Normalizacja to transformacja dokumentów w korpusie, tak żeby tekst był zapisany w formacie standardowym. Najprostsza metoda normalizacji to zamiana w tekście wielkich liter na małe.

LLM to wielki model językowy, czyli wytrenowany wstępnie model, który posiada wiedzę na temat relacji tekstowych.

2.2 Zadania NLP

Obliczeniowy charakter przetwarzania języka naturalnego może zostać scharakteryzowany przez zadania, których celem jest analiza testu pod kątem ekstrakcji pewnych cech. Jak zauważają Siebers

et al., zadania te tworzą hierarchię, w której trudniejsze zadanie jest oparte na prostszym [70]. Zadaniami tymi są:

Analiza leksykalna, która jest najbardziej podstawowym poziomem analizy tekstu. Polega na badaniu struktury słów, co uwzględnia ich ortografię, morfologię (czyli ich budowę i odmianę), a także w przypadku tekstu mówionego - fonologię. Centralnym zadaniem tej analizy jest tokenizacja tekstu.

Analiza syntaktyczna rozszerza analizę leksykalną – z analizy poszczególnych słów do całych zdań – w celu analizy zasad gramatycznych. Głównym zadaniem analizy syntaktycznej jest oznaczanie części mowy w tekście (*part-of-speech* - POS), w zależności od ich roli gramatycznej. Jednymi z rozważanych ról gramatycznych są: czasownik, rzeczownik, przyniomość i inne.

Analiza semantyczna bada znaczenie słów i zdań. Liczba zadań, którymi zajmuje się analiza semantyczna jest większa od poprzednich analiz, a należą do niej między innymi: klasyfikacja tekstu, tłumaczenie maszynowe, podsumowywanie tekstu, itp...

Analiza pragmatyczna zajmuje się odkrywaniem niedosłownego sensu zdań, a także rozróżnianiem tego co jest powiedziane od tego co faktycznie zostało przekazane. Jednym z zadań jest rozpoznawanie relacji hiponimii, która jest porządkującą relacją hierarchiczną, która zachodzi, gdy znaczenie danego bardziej szczegółowego słowa zawiera się w innym bardziej ogólnym słowie, np. słowa „gołąb”, „kruk” i „wróbel” zawierają się w jednym słowie „ptak” – wszystkie oznaczają ptaka [1].

2.3 Struktury danych

Aby zrozumieć dalsze informacje, to niezbędne jest ustalenie znaczenia użytych struktur danych.

Skalar jest prostą wartością liczbową, np. 15, -3.25 . **Wektor** jest uporządkowaną listą wartości skalarnych, które oznaczają w niej wartości cech. Wektory mogą być zwizualizowane jako skierowane w pewnym kierunku strzałki oraz jako punkt w wielowymiarowej przestrzeni. **Macierz** to prostokątna tablica liczb, ułożona w rzędy i kolumny. Można ją interpretować jako strukturę złożoną z wektorów, które mogą reprezentować rzędy lub kolumny. **Tensor**, to uogólnienie macierzy w wyższych wymiarach [7]. W PyTorchu tensor może reprezentować skalar, wektor, macierz lub właśnie uogólnienie macierzy na więcej wymiarów niż 2 [26].

2.4 Uczenie maszynowe

Według Burkova [7], uczenie maszynowe to dziedzina informatyki skupiona na nadaniu programom komputerowym umiejętności do automatycznego dostrojenia swojego zachowania w zależności od dostarczonych obserwacji. Obserwacje są przeważnie dostarczane w formie **zbioru danych** lub próbki pomiarów, i mogą pochodzić ze świata rzeczywistego lub z wyniku działania innego algorytmu. Dostrajane parametry są częścią **statystycznego modelu**, który rozwiązuje dany problem.

Wartości wprowadzane do modelu są nazywane **cechami**. Może nimi być wartość pobierana ze zbioru danych z odpowiedniej kolumny, przez co można zamiennie mówić cecha i kolumna.

Mozliwe rodzaje uczenia maszynowego:

- **Nadzorowane** - wykorzystuje się **oznaczone przykłady** $\{(\vec{x}_1, \vec{y}_1), (\vec{x}_2, \vec{y}_2), \dots, (\vec{x}_N, \vec{y}_N)\}$. Każdy element \vec{x}_i z pośród N jest nazywany **wektorem cech**. Każda wartość $(v_1, v_2, \dots, v_M) \in$

\vec{x}_i tego wektora jest nazywana cechą. Algorytm treningowy dostarcza modelowi wektor \vec{x}_i i dostraja jego parametry na podstawie wektora \vec{y}_i oraz wyniku działania modelu \vec{y}_i .

- **Częściowo nadzorowane** - wykorzystuje się zarówno **oznaczone przykłady** jak i **nienadzorowane przykłady**. Liczba nieoznaczonych przykładów jest przeważnie większa niż oznaczonych przykładów. Cel tego rodzaju uczenia jest taki sam jak w uczeniu nadzorowanym, a jednocześnie wykorzystuje się go z nadzieją, że użycie wielu nieoznaczonych przykładów, pomoże algorytmowi dostroić model tak, aby był lepszy niż z wykorzystaniem samych oznaczonych przykładów.
- **Nienadzorowane** - wykorzystuje **nienadzorowane przykłady** $\{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N\}$. Jego celem jest stworzenie modelu, który otrzymuje wektor cech \vec{x}_i i transformuje go albo w inny wektor albo wartość, którą można wykorzystać do rozwiązyania jakiegoś zadania. Ta metoda może być wykorzystana do **grupowania, redukcji wymiarów lub detekcji odstających przykładów**.
- **Ze wzmacnieniem** - polega na nauczeniu optymalnej **strategii** w środowisku, w którym istnieje **agent**, który jest sterowany przez decyzje podejmowane przez model. Agent otrzymuje stan środowiska, w którym się znajduje w postaci wektora i może podejmować decyzje, które zmieniają jego stan w sposób, który nie musi kończyć działania procesu, czyli podejmuje je w jego trakcie, a nie po zakończeniu działania programu. Akcje, które podejmuje generują **nagrodę**. Optymalna strategia jest funkcją, która przyjmuje wektor stanu środowiska i zwraca optymalną akcję do wykonania w tym stanie. Celem uczenia jest stworzenie takiej strategii, która maksymalizuje nagrodę w dłuższej perspektywie.

Algorytmy uczenia maszynowego rozwiązują między innymi problemy regresji, klasyfikacji, grupowania i redukcji wymiarów.

Regresja polega na takim dostrojeniu parametrów modelu, aby funkcjonował jako funkcja, która jest w stanie przewidzieć jakąś wartość na podstawie rzeczywistych obserwacji, np. poziom wypłaty pracownika, biorąc pod uwagę jego doświadczenie oraz wiedzę.

Klasyfikacja skupia się na przewidywaniu właściwej klasy ze zbioru możliwych klas dla jakiegoś zjawiska, np. zaklasyfikowanie wypowiedzi do jednej z możliwych klas prawdziwości w serwisie fact checkingowym.

Grupowanie pozwala jak najlepiej podzielić jakiś zbiór przykładów, w taki sposób, aby powstały grupy mające podobne cechy. W tej metodzie model zwraca identyfikator grupy, do której przynależy dany przykład, a algorytm trenujący próbuje znaleźć jak najlepsze parametry, aby grupy spełniały określone własności.

Redukcja wymiarów jest istotna, ponieważ pozwala zmniejszyć wymagania obliczeniowe modelu poprzez transformację większego tensora na mniejszy, przy zachowaniu pewnych informacji. Ale przede wszystkim jest przydatna do lepszego zrozumienia danych. Pozwala między innymi wizualizować wektory cech, które mają więcej niż trzy wymiary, w przestrzeni o mniejszej liczbie wymiarów, np. 3D.

2.5 Sieci neuronowe

2.5.1 Optymalizator

Algorytm spadku gradientowego (z ang. *Gradient Descent*) jest jednym z wielu algorytmów wykorzystywanych do uczenia modeli sieci neuronowych. Bazuje on na gradiencie funkcji straty

$\nabla_{\theta_i} L(\theta_i, y, \hat{y})$ do iteracyjnego obliczania zmian parametrów wag we wszystkich warstwach, zaczynając od ostatniej i przekazując gradient wstecz, do poprzednich warstw, co nazywa się mechanizmem **backpropagation**.

W procesie uczenia algorytm optymalizuje każdą wagę θ_i przy pomocy wzoru:

$$\theta_i^{(t)} = \theta_i^{(t-1)} - \eta \nabla_{\theta_i} L(\theta_i^{(t-1)}, y, \hat{y})$$

, gdzie:

- t jest krokiem iteracji.
- y reprezentuje oczekiwane wyjście.
- \hat{y} reprezentuje wartość przewidywaną przez model.
- η jest współczynnikiem uczenia (z ang. *learning rate*)

Wszystkie parametry są aktualizowane po przetworzeniu jednej partii danych, zwanej *batch*, która zawiera wszystkie przykłady ze zbioru treningowego. To podejście jednak nie jest popularnie wykorzystywane ze względu na wysoki koszt pamięciowy i czasowy, gdyż wagi są aktualizowane dopiero po przetworzeniu wszystkich przykładów, których mogą być miliony. Zamiast tego stosuje się stochastyczne metody takie jak *Stochastic Gradient Descent* (SGD), jego wersję z momentem (*momentum*), itp... SGD wykorzystuje mniejsze partie danych, które są podzbiorami całego zbioru danych, które nazywa się *mini-batch*. W literaturze naukowej często używa się pojęć *batch* i *mini-batch* zamiennie w kontekście modeli stosujących mini-batche. W kolejnych rozdziałach również stosujemy te pojęcia zamiennie, ale zawsze mamy na myśli *mini-batch*.

Stochastyczność [15] aktualizacji wag oraz dobór hiperparametru η powoduje, że trudniej jest trafić w globalne optimum lub dobre lokalne minimum, ponieważ podstawowy algorytm SGD ze zbyt wysokim η może je przeskakiwać, a ze zbyt niskim zabiera zbyt dużo czasu, żeby do nich trafić, przez co wymaga ręcznego dostrojenia η , co jest czasowo kosztowne. Zastosowanie momentów przeciwdziała problemowi wolnej konwergencji i oscylacji poprzez uwzględnianie poprzednich kierunków aktualizacji, co wygładza trajektorię spadku i pozwala szybciej osiągnąć minimum. Algorytm zapamiętuje poprzedni kierunek zmiany wag i dostraja je osobno do ich momentu poprzez dostosowanie wielkości kroku. Każda waga jest aktualizowana na podstawie własnej historii gradientów (momentu). Podstawowy wzór aktualizacji wag, uwzględniający moment to:

$$\begin{aligned} v_i^{(t)} &= \mu v_i^{(t-1)} - \eta \nabla_{\theta_i} L(\theta_i^{(t-1)}, y, \hat{y}) \\ \theta_i^{(t)} &= \theta_i^{(t-1)} + v_i^{(t)} \end{aligned}$$

, gdzie:

- $v_i^{(t)}$ - jest momentem w aktualnym kroku t (początkowo ma wartość 0).
- μ - jest współczynnikiem tłumiącym moment (zwykle przybiera wartość 0.9 i przypomina współczynnik tarcia w fizyce).

W Adagrad stosuje się adaptacyjne dopasowywanie η poprzez wzór:

$$c_i^{(t)} = c_i^{(t-1)} + (\nabla_{\theta_i} L(\theta_i^{(t-1)}, y, \hat{y}))^2$$

$$\theta_i^{(t)} = \theta_i^{(t-1)} - \eta \nabla_{\theta_i} L(\theta_i^{(t-1)}, y, \hat{y}) / (\sqrt{c_i^{(t)}} + \epsilon)$$

, gdzie:

- $c_i^{(t)}$ - jest sumą poprzednich kwadratów gradientu w kroku t .
- ϵ - jest małą stałą zapobiegającą dzieleniu przez zero (zwykle $1e-8$).

Gdy gradient wagi jest duży, to waga jest aktualizowana z mniejszym krokiem, a gdy jest mały to z większym krokiem. Ponieważ z czasem gradienty stają się bardzo małe, a suma kwadratów gradientów zawsze rośnie, to po pewnej liczbie iteracji, wartości aktualizacji wag stają się bardzo niskie.

Aby temu przeciwdziałać RMSProp stosuje dostosowanie η zaimplementowane z wykorzystaniem *Exponentially Weighted Average* (EWMA) na gradiencie, zamiast wykorzystywać czysty gradient. W SGD z momentami również można stosować to podejście [8]. Uogólniony wzór na EWMA:

$$V_t = \beta V_{t-1} + (1 - \beta)S_t$$

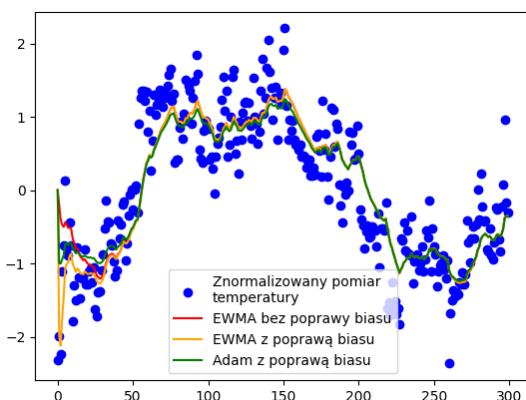
$$\beta \in [0, 1]$$

, gdzie:

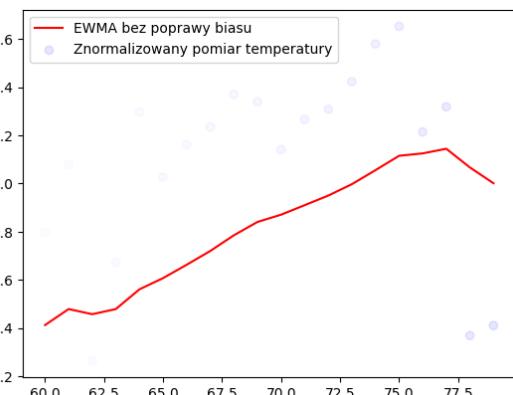
- V_t - aktualna średnia wartość.
- β - jest parametrem, który steruje poziomem udziału skumulowanej średniej (zwykle stosuje się wartość 0.9).
- S_t - wartość jakiegoś pomiaru.

Dzięki zastosowaniu EWMA uwzględniany jest udział gradientów z najbliższych poprzedzających iteracji, a gradienty z dalszych iteracji są wykładniczo mniej uwzględniane. W pewien sposób są zapominane. Przykład działania EWMA można zobaczyć na Rysunkach 2.1 i 2.2.

RYSUNEK 2.1: Przykład działania EWMA na małym zbiorze wartości zmierzonych średnich temperatur w Warszawie [32]. Adam jest liczony dla *amsgrad* \leftarrow *False*, zamiast $g^{(t)}$ wykorzystywane są pomiary temperatury. Pomiary temperatury są znormalizowane poprzez StandardScaler z paczki *sklearn.preprocessing*.



RYSUNEK 2.2: Zobrazowanie wpływu poprzedzających pomiarów od punktu nr 80 w wycinku od punktu nr 60 do 80. Jasność pomiarów temperatury pokazuje wpływ jego wagi na aktualną wartość średniej. Wykres odnosi się do oryginalnego wykresu na Rysunku 2.1.



Bardziej zaawansowaną i szerokostosowaną metodą stochastyczną jest Adam (*Adaptive Moment Estimation*) [47, 23]. Do dostrajania wag wykorzystuje pierwszy i drugi moment, które są sterowane kolejno hiperparametrami β_1 i β_2 (zwykle przyjmują kolejno wartości 0.9 i 0.99). Istnieje również hiperparametr ϵ , który steruje małą wartością, potrzebną do zapobiegnięcia dzielenia przez zero w

Algorithm 1 Algorytm Adam używany w PyTorch [23]

```

for t = 1 to ... do
    if maximize then
         $g^{(t)} \leftarrow -\nabla_{\theta} f^{(t)}(\theta^{(t-1)})$ 
    else
         $g^{(t)} \leftarrow \nabla_{\theta} f^{(t)}(\theta^{(t-1)})$ 
    end if
end for
if  $\lambda \neq 0$  then
     $g^{(t)} \leftarrow g^{(t)} + \lambda \theta^{(t-1)}$  (Uwzględnienie regularyzacji L2)
end if
 $m^{(t)} \leftarrow \beta_1 m^{(t-1)} + (1 - \beta_1)g^{(t)}$  (Aktualizacja estymacji pierwszego momentu z biasem)
 $v^{(t)} \leftarrow \beta_2 v^{(t-1)} + (1 - \beta_2)(g^{(t)})^2$  (Aktualizacja estymacji drugiego momentu z biasem)
 $\hat{m}^{(t)} \leftarrow m^{(t)} / (1 - \beta_1^t)$  (Obliczenie poprawionej o bias estymacji wartości pierwszego momentu)
if amsgrad then
     $v_{max}^{(t)} \leftarrow \max(v_{max}^{(t-1)}, v^{(t)})$ 
     $\hat{v}^{(t)} \leftarrow v_{max}^{(t)} / (1 - \beta_2^t)$ 
else
     $\hat{v}^{(t)} \leftarrow v^{(t)} / (1 - \beta_2^t)$  (Obliczenie poprawionej o bias estymacji wartości drugiego momentu)
end if
 $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta \hat{m}^{(t)} / (\sqrt{\hat{v}^{(t)}} + \epsilon)$  (Aktualizacja parametru)

```

ostatnim wzorze na aktualizację wagi (zwykle przyjmuje wartość 1e-8). Implementacja PyTorch jest przedstawiona w Algorytmie 1.

Aby zapobiegać przeuczeniu, to w głębokim uczeniu maszynowym stosuje się regularyzację L2, która przeciwdziała rosnącym wartościom wag. Wysokie wartości wag powodują, że małe zmiany w wartościach wejściowych znacznie wpływają na zmiany aktywacji neuronów, co negatywnie wpływa na generalizację modelu.

Ze względu na to, że każda waga jest efektywnie dostrajana przy pomocy własnej wyliczanej wartości η , to różne wagi mogą otrzymywać różne wartości regularyzacji L2 i aby temu zapobiegać, to stosuje się inną wersję tego algorytmu – AdamW [53, 24], która stosuje *weight decay*, który jest równoznaczny regularyzacji L2 przy zastosowaniu SGD bez momentów, ale nie w adaptatywnych algorytmach gradientowych. Weight decay jest nakładane osobno na wagi w oderwaniu od funkcji straty, w której przeważnie uwzględnia się współczynnik regularyzacji L2, i dzięki temu lepiej sprawdza się jako tego typu metoda regularyzacji w optymalizatorze Adam.

2.5.2 Funkcje aktywacji

Rectified Linear Unit (ReLU) [28] jest popularną funkcją aktywacji zdefiniowaną jako:

$$\text{ReLU}(x) = \max(0, x)$$

Wprowadza nieliniowość i posiada stosunkowo prostą pochodną, dzięki czemu ma niski koszt obliczeniowy, co jest pożądane w algorytmie wstępnej propagacji. Pochodna ReLU jest dana wzorem:

$$\text{ReLU}'(x) = \begin{cases} 0, & \text{gdy } x \leq 0 \\ 1, & \text{gdy } x > 0 \end{cases}$$

Funkcja ReLU lepiej oddaje nieliniowy charakter aktywacji neuronów biologicznych w porównaniu do funkcji Sigmoid lub Tanh, a także sprzyja wprowadzeniu rzadkości aktywacji w modelu. Rzadkie reprezentacje charakteryzują się:

- Uproszczeniem interpretacji relacji pomiędzy wejściem a wyjściem — w gęstych reprezentacjach rozproszonych zmiana jednej składowej wejścia często wpływa na większość składowych wyjścia, natomiast rzadkie reprezentacje umożliwiają uzyskanie bardziej lokalnych i stabilnych aktywacji, ułatwiając powiązanie cech wejściowych z konkretnymi elementami reprezentacji (czyli cechami wyjściowymi).
- Selektyną aktywacją neuronów — w danym przykładzie tylko część neuronów jest aktywna, co wprowadza zróżnicowanie rozmiaru efektywnej reprezentacji w warstwach.
- Łatwiejszym liniowym oddzieleniem reprezentacji, co sprzyja skutecznieszemu uczeniu.

Ze względu na zerową pochodną ReLU dla wartości ujemnych pojawia się problem tzw. *dying ReLU*. Objawia się on sytuacją, w której neuron stale zwraca wartość zero (nie jest aktywowany), przez co gradient dla jego wag wynosi zero i wagi nie ulegają aktualizacji. Jeśli dla wszystkich przykładów w zbiorze danych neuron pozostaje nieaktywny, nie jest w stanie „ożyć” ponownie, co prowadzi do utraty jego zdolności uczenia. Z tego powodu zbyt duża liczba takich neuronów negatywnie wpływa na zdolność przewidywania, bo zmniejsza efektywną pojemność informacyjną modelu.

Gaussian Error Linear Unit (GELU) [35] można interpretować jako mnożenie wejścia x przez prawdopodobieństwo, że zmienna losowa $X \sim \mathcal{N}(0, 1)$ przyjmie wartość mniejszą lub równą x . W oryginalnej pracy opisano to jako mnożenie przez zmienną losową o rozkładzie Bernoulliego, z parametrem $\Phi(x) = P(X \leq x)$, jednak w praktycznej implementacji stosuje się jej wartość oczekiwana, co prowadzi do deterministycznej postaci:

$$\text{GELU}(x) = x \Phi(x),$$

gdzie $\Phi(x)$ jest dystrybuantą standardowego rozkładu normalnego:

$$\Phi(x) = P(X \leq x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} e^{-u^2/2} du.$$

Dystrybuantę $\Phi(x)$ można obliczyć przy pomocy funkcji błędu $\text{erf}(x)$:

$$\text{GELU}(x) = x \Phi(x) = \frac{x}{2} \left(1 + \text{erf} \left(\frac{x}{\sqrt{2}} \right) \right).$$

Ze względu na koszt obliczeniowy w praktyce stosuje się przybliżenia, np.:

$$\text{GELU}(x) \approx 0.5x \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + 0.044715x^3) \right) \right),$$

lub

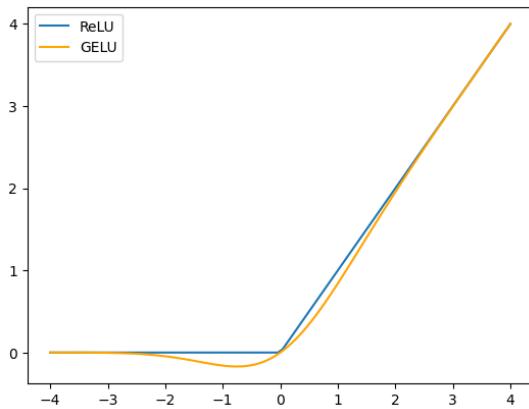
$$\text{GELU}(x) \approx x \sigma(1.702x),$$

gdzie $\sigma(\cdot)$ oznacza funkcję sigmoidalną.

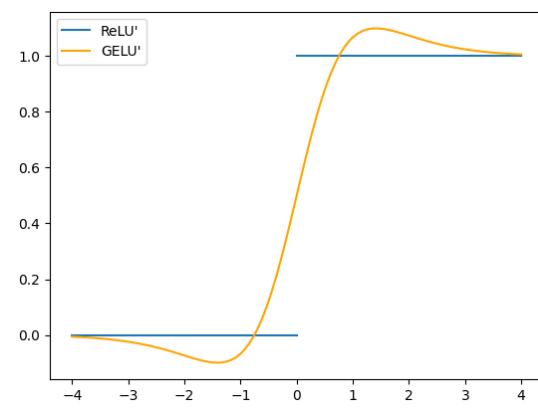
GELU zapobiega problemowi „dying ReLU”, ponieważ jest funkcją gładką i niemonotoniczną, której wartość może zarówno rosnąć, jak i maleć wraz ze wzrostem x . Małe ujemne wartości wejściowe dają niewielkie ujemne aktywacje, co zapewnia niezerowy gradient również w tym zakresie, w przeciwieństwie do ReLU.

Funkcja zachowuje nielinowość i dzięki cechom niewypukłości oraz niemonotoniczności jest w stanie lepiej przybliżać skomplikowane funkcje w porównaniu do ReLU, która jest wypukła i monotoniczna. Na Rysunkach 2.3 i 2.4 przedstawiono wykresy oraz pochodne obu funkcji aktywacji dla lepszego zobrazowania różnic.

RYSUNEK 2.3: Porównanie funkcji aktywacji ReLU i GELU



RYSUNEK 2.4: Porównanie pochodnych funkcji aktywacji ReLU i GELU



Stochastyczną maskę $\Phi(x)$, opartą na rozkładzie normalnym, wybrano spośród innych wariantów, ponieważ wejścia neuronów często mają rozkład zbliżony do normalnego, zwłaszcza przy zastosowaniu metod normalizacji. BERT i RoBERTa stosują funkcję aktywacji GELU.

2.5.3 Dropout

Głębokie sieci neuronowe są w stanie bardzo dobrze dopasować parametry tak, aby odwzorować prawie każdą funkcję. Niestety to niesie za sobą ryzyko przeuczenia, które polega na zbytnim dostosowaniu się modelu do przykładów i szumu zbioru treningowego, przez co radzi sobie gorzej z danymi, których nie widział w procesie trenowania. Jedną z metod przeciwdziałania przeuczeniu jest **Dropout**, który polega na losowym „wyłączaniu” neuronów (zerowaniu wartości neuronów) podczas trenowania modelu [7, 64, 6]. Stosuje się go na wyznaczonych warstwach (przeważnie w warstwach ukrytych). Przyjmuje parametr p , który steruje prawdopodobieństwem wyzerowania neuronu. Gdy $p = 0$, to dropout nie jest nakładany, a gdy $p = 1$, to wszystkie neurony są wyzerowywane. Wartość parametru p można ustalić eksperymentalnie.

2.5.4 Softmax

Softmax to funkcja, która jest powszechnie stosowana w wyjściowych warstwach sieci neuronowych, szczególnie w zadaniach klasyfikacji wieloklasowej [6]. Jej celem jest przekształcenie wektora logitów w rozkład prawdopodobieństwa. Dla wektora \mathbf{z} :

$$\mathbf{z} = [z_1, z_2, \dots, z_k]$$

, gdzie k to liczba klas decyzyjnych, funkcja softmax jest definiowana jako:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$$

Składowe wyników funkcji softmax zawsze mieszczą się w przedziale $[0, 1]$ i sumują się do 1. Softmax nie zmienia porządku wartości wejściowych, a jej wynik można interpretować jako prawdopodobieństwo przynależności danej próbki do danej klasy.

2.6 Przekleństwo wymiarowości i błogosławieństwo nierównomierności

Przekleństwo wymiarowości jest zjawiskiem, w którym wraz ze wzrostem liczby cech (wymiarów), przestrzeń możliwych danych rośnie tak szybko, że aby zachować podobną gęstość przy-

kładów, trzeba posiadać ich wykładowiczo więcej. Spełnienie tego wymogu jest kluczowe, ponieważ efektywność modelu jest znacznie niższa, gdy zostanie wytrenowany na zbyt małym pokryciu możliwej efektywnej przestrzeni przykładów. W wysokich wymiarach punkty stają się od siebie bardzo odległe, a większość przestrzeni „przesuwa się” ku jej granicom, co utrudnia modelom skuteczne uczenie się [19].

Jednocześnie warto zauważyc, że większość problemów nie ma równomiernie rozproszonych przykładów w przestrzeni. Przykładowo, pewien model może sobie dobrze poradzić z klasyfikacją ręcznie narysowanych cyfr, pomimo że obrazy cyfr mają po jeden wymiar na piksel. Jest tak ponieważ **efektywna przestrzeń** możliwych obrazów cyfr jest mniejsza niż przestrzeń wszystkich jednorodnie rozproszonych punktów i jest pewien sposób skoncentrowana. To zjawisko nazywa się **błogosławieństwem nierównomierności**.

2.7 Reprezentacja tekstu

Nowoczesne modele językowe są oparte o sieci neuronowe, które działają dzięki obliczeniom prowadzonym na liczbach, a nie na symbolach, które są dyskretnymi jednostkami informacji składającymi się na słowa. Z tego powodu tekst w przetwarzaniu języka naturalnego wymaga odpowiedniej reprezentacji zdolnej do efektywnego przetwarzania przez sieć neuronową.

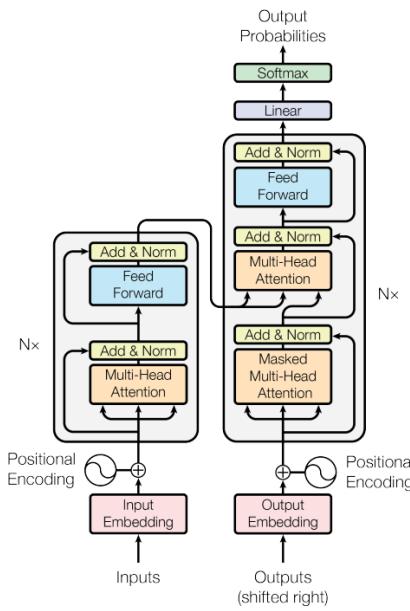
Na potrzeby zrozumienia dalszego opisu warto zaznaczyć, że znaczenie symbolu w NLP nie jest równoznaczne ze znaczeniem tokenu. Symbole nie posiadają relacji językowych, poza relacją kolejności w odpowiednim dla nich słowniku (np. w tabeli ASCII).

Autorzy pracy „A Survey of Text Representation Methods and their Genealogy” [70] opisali, w swoim przeglądzie literatury, różne metody reprezentacji tekstu oraz ich rozwój historyczny. Rozróżniają dwie podstawowe metody:

Lokalne reprezentacje tekstu tłumaczą postać tokenową na postać wektora liczb. W najprostszym ujęciu wykorzystuje się *one-hot encoding*, który zamienia token, a konkretnie pojedyncze słowo, na wektor rzadki, gdzie wartość równa 1, dla składowej na odpowiedniej pozycji, reprezentuje konkretny token, a wszystkie inne składowe wektora są równe 0. W ten sposób każdy wektor może unikalnie reprezentować dany token. Dzięki temu wektory są łatwo interpretowalne, ale brakuje możliwości analizy podobieństwa semantycznego, ponieważ wszystkie wektory w słowniku są względem siebie ortogonalne i mają równy dystans względem wszystkich innych. Istnieje również reprezentacja *bag-of-words* (BOW), która polega na zliczeniu (zsumowaniu) wektorów one-hot z danego dokumentu, aby uzyskać informację o częstotliwości występowania tokenów, dzięki czemu porównując takie reprezentacje, można wnioskować na temat podobieństwa dokumentów.

Dystrybucyjne reprezentacje tekstu również tłumaczą postać tokenową na postać wektora liczb, ale robią to z uwzględnieniem *hipotezy dystrybucyjności*, która stanowi, że znaczenie słowa jest określane przez rozkład jego sąsiednich słów, czyli informacji o współwystępowaniu [33, 70]. Przykładowo słowa „widelec” i „nóż” mogą występować w sąsiedztwie w wielu dokumentach, dzięki czemu ich wektory będą reprezentować punkty w przestrzeni, które są blisko siebie, a słowa „widelec” i „szampon” będą rzadziej współwystępować, przez co ich pozycje będą dalej od siebie niż pozycje słów „widelec” i „nóż”. Dodatkowo słowa znaczące to samo, ale mające inną odmienioną formę leksykalną będą jeszcze bliżej siebie niż słowa o podobnym znaczeniu semantycznym. Mikolov et al. [54, 70] pokazują, że operacja na wektorach $\text{vector}(\text{``king''}) - \text{vector}(\text{``man''}) + \text{vector}(\text{``woman''})$ na wytrenowanych embeddingach jest najbliższa wektorowi $\text{vector}(\text{``queen''})$, ukazując że dostarczają one informacji semantycznych. Wektory one-hot o rozmiarze M są znacznie większe niż wektory w tej reprezentacji (o rozmiarze N), ponieważ M jest równe wielkości używanego słownika, a N jest pewną mniejszą, stałą wartością, równą przykładowo 768 składowych. **Embeddingi** są

RYSUNEK 2.5: Architektura Transformera przedstawiona w artykule „Attention Is All You Need” [76]. Lewa strona reprezentuje koder, a prawa to dekoder.



synonimem reprezentacji dystrybucyjnych. Warto pamiętać, że zakodowanie tokenu w postaci embeddingu wymaga macierzy o rozmiarze $M \times N$, co w przypadku pojedynczych tokenów może być bardziej pamięciochłonne niż wektory one-hot. Jednak przy przetwarzaniu dużej liczby przykładów zastosowanie embeddingów wraz z macierzą projekcji staje się rozwiązaniem bardziej efektywnym pamięciowo. Dystrybucyjne reprezentacje tekstu, np. *word2vec*, który pozwala reprezentować pojedyncze tokeny, są trudne w interpretacji, ponieważ abstrakcyjne i złożone koncepcje językowe są ogólnie rozłożone na wszystkich składowych wektora, a dodatkowo wiele składowych uczestniczy w opisywaniu wielu koncepcji na raz.

Nie wszystkie reprezentacje dystrybucyjne kodują informację kontekstową (np. nie robią tego embeddingi tokenów *word2vec*), dlatego stosuje się bardziej zaawansowane metody, który tworzą reprezentacje dystrybucyjne, które uwzględniają różne zadania NLP, jak np. uwzględnienie kontekstu.

2.8 Architektura Transformer

Architektura *Transformer* [76] to sieć neuronowa służąca do zamiany jednego tekstu na inny (np. w procesie tłumaczenia maszynowego – z tekstu źródłowego do wybranego języka). Schemat architektury jest przedstawiony na Rysunku 2.5.

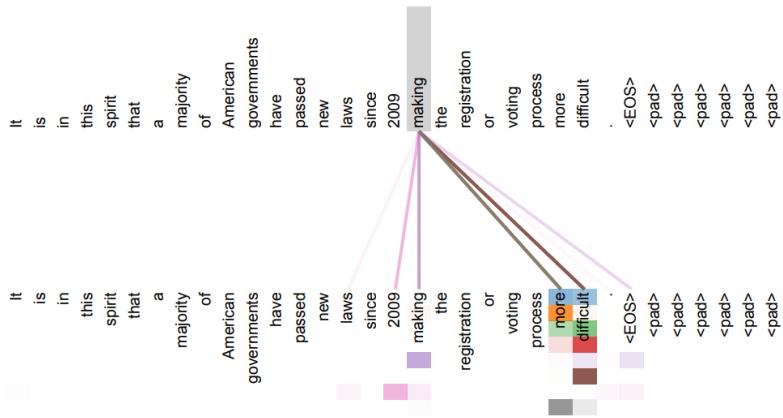
Składa się z dwóch modułów:

- **kodera**, którego rolą jest transformacja tekstu w postaci sekwencji tokenów, do postaci sekwencji wektorów opisujących znaczenie tych tokenów, z uwzględnieniem kontekstu,
- **dekodera**, którego rolą jest skonsumowanie wektorów z kodera i wygenerowanie wyjściowej sekwencji tokenów (np. przetłumaczzonego tekstu).

Zarówno koder jak i dekoder składają się z bloków, z których każdy posiada:

- mechanizm atencji, starający się wyrazić znaczenie tokenu jako wypadkową wszystkich tokenów w sekwencji,

RYSUNEK 2.6: Przykład obrazujący zachowanie mechanizmu atencji w Transformerze. Przedstawiony rysunek ukazuje zależności pomiędzy słowem „making” a innymi odległymi słowami. Różne głowy atencji są zaznaczone różnymi kolorami. Wybrane zostały wartości z warstwy 5. i 6.



- warstwę połączeń każdy z każdym (*Feed Forward Neural Network*) służącą do transformacji wektorów do nowej postaci,
- połączenia rezydualne poprawiające jakość uczenia się przy głębszej sieci neuronowej,
- warstwy normalizacyjne, zapewniające, że wyjścia każdego bloku będą miały niewielkie wartości dookoła zera

Ponieważ mechanizm atencji nie uwzględnia automatycznie kolejności występowania tokenów – wyraża znaczenie tokenu jako średnią ważoną wszystkich tokenów w sekwencji, do reprezentacji wejściowych dodajemy wektory kodujące pozycję (tzw. *positional embeddings*), które pomagają sieci zrozumieć na jakiej pozycji występuje dany token.

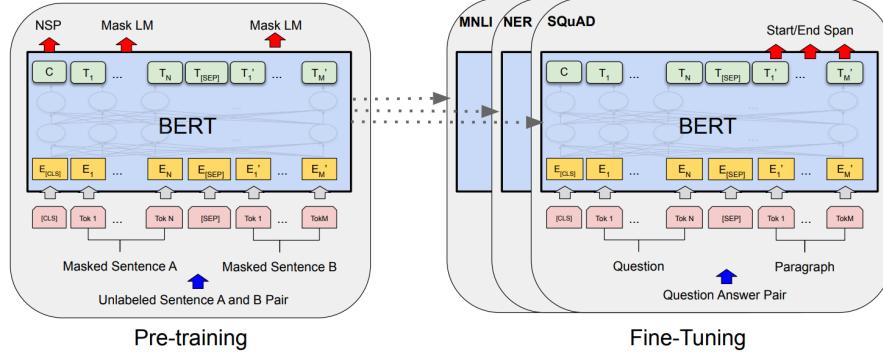
Atencja to taki mechanizm, który porównuje znaczenie danego tokenu ze wszystkimi dostępnymi tokenami, liczy podobieństwo zgodnie z wybraną miarą i konstruuje nową reprezentację generując sumę ważoną po reprezentacjach zgodnie z podobieństwami. W Transformerze ten mechanizm jest zaimplementowany w bloku *Multi-Head Attention*, który zawiera wiele głów atencji (*self-attention head*), gdzie każda z nich oblicza inny rodzaj relacji pomiędzy tokenami, a następnie wygenerowane reprezentacje są scalane w jedną reprezentację wynikową.

2.8.1 Model BERT

BERT (*Bidirectional Encoder Representations from Transformers*) [17] to model językowy służący do generowania reprezentacji dystrybucyjnych tekstu. Jest oparty na architekturze Transformerze, a precyzyjniej na jego koderze. Koduje tekst dwukierunkowo, uwzględniając w każdym tokenie jego relacje względem tokenów z lewej oraz z prawej, dzięki zastosowaniu niemaskowanego mechanizmu *Multi-Head Attention* - przez co uwzględnia w tokenach kontekst zdania. Model przetwarza sekwencje tokenów, które mogą reprezentować pojedynczy tekst (np. zdanie), ale również wiele tekstów (np. pytania i odpowiedzi, pary zdań do sklasyfikowania, itp.), jeśli te teksty te oddzielone są specjalnym znacznikiem separatora [SEP] informującym model, że tekst z lewej strony separatora jest czymś innym niż z prawej strony separatora.

Pierwszy token wejścia to specjalny token [CLS], następnie są tokeny ciągu A oraz ewentualnie B, oddzielone od siebie (w sensie pary) tokenem separatora [SEP] (ostatni token sekwencji wejściowej również jest tokenem /SEP/). Embeddingi wejściowe są dodatkowo modyfikowane poprzez dodanie do nich wyuczonych embeddingów kodujących przynależność do zdania A lub B.

RYSUNEK 2.7: Schemat modelu BERT w procesie inicjalnego treningu i fine-tuningu. Po lewej stronie można zobaczyć, które tokeny są wykorzystywane do zadania NSP oraz Masked LM (MLM), a także obrazuje jak wygląda podział na zdanie A i B. Po prawej widać schemat zastosowania modelu zadaniach docelowych, z wykorzystaniem fine-tuningu. SQuAD reprezentuje zadanie Question Answering (QA). NER to Named Entity Recognition. MNLI to zadanie predykcji zaprzeczenia, potwierdzenia albo neutralności zdania B w stosunku do zdania A, które dotyczy benchmarku GLUE [17].



Model jest w stanie efektywnie kodować sekwencje wejściowe dzięki inicjalnemu treningowi wstępemu na ogromnych zbiorach tekstu z BooksCorpus (zawierający 800 milionów słów [82]) oraz English Wikipedia (zawierająca 2.5 miliona słów), podczas którego model jest dostosowywany do rozwiązywania dwóch zadań: Masked LM (MLM) oraz Next Sentence Prediction (NSP).

Masked LM polega na wyborze w każdym przykładzie, podczas przygotowywania zbioru, 15% tokenów osobno w zdaniu A oraz w zdaniu B, dla których 80% zostanie podmienione na specjalny token [MASK], 10% zostanie podmienione na inny losowy token, 10% zostanie niezmienione. Celem modelu jest przewidzieć poprawnie identyfikatory oryginalnych tokenów z używanego słownika, które zostały zamaskowane tokenem [MASK], jedynie na podstawie kontekstu, czyli tokenów sąsiednich słów. Dzięki maskowanemu modelowi językowemu, symuluujemy zadanie zgadywania zasłoniętego słowa, aby dobrze zgadnąć słowo, musimy zrozumieć kontekst. Rezultatem tego zadania jest wyuczenie się obustronnych relacji pomiędzy słowami w zdaniach.

Next Sentence Prediction wykorzystuje specjalny token [CLS], trenowany w taki sposób, aby kodował zbiorczą reprezentację całego wejścia. Celem zadania jest predykcja, czy zdanie B jest następującym zdaniem po A. Wykorzystywany zbiór zawiera 50% przypadków, gdy zdanie A poprzedza zdanie B i 50% przypadków, w których tak nie jest. Rezultatem zadania ma być wyuczenie się relacji pomiędzy dwoma zdaniami, której nie można się nauczyć jedynie poprzez zadanie MLM.

Autorzy definiują liczbę bloków kodera jako L , długość wektora ukrytego jako H , a także liczbę głów self-attention jako A . Prezentowane są wyniki dla dwóch wielkości modeli $BERT_{BASE}$ ($L=12$, $H=768$, $A=12$, całkowita liczba parametrów to 110 milionów) i $BERT_{LARGE}$ ($L=24$, $H=1024$, $A=16$, całkowita liczba parametrów to 340 milionów). Pokazane jest, że im większy rozmiar modelu, tym lepsze wyniki osiąga na zbiorze deweloperskim, np. gdy ($L=24$, $H=1024$, $A=16$), to w zadaniach MNLI-m, MRPC oraz SST-2 z benchmarku modeli językowych GLUE [77] osiąga kolejno wyniki: 86.6, 87.8, 93.7; a gdy ($L=12$, $H=1024$, $A=16$), to osiąga kolejno niższe wyniki: 85.7, 86.9, 93.3.

Twórcy prezentują, że tak przygotowany model można następnie łatwo dostroić zarówno poprzez metody *fine-tuning* oraz *feature-based*. Fine-tuning polega na dostrojeniu wag całego modelu, włącznie z przetrenowanym wcześniej modelem BERT oraz dodatkowymi warstwami służącymi do rozwiązywania zadania docelowego (*downstream task*), na mniejszym zbiorze, co zabiera mniej czasu niż wytrenowanie modelu od zera.

Metoda feature-based jest mniej inwazyjna, ponieważ polega na wykorzystaniu przygotowanych

wcześniej reprezentacji wygenerowanych przez wstępnie wytrenowany model BERT i wytrenowaniu jedynie dodatkowych warstw do zadania docelowego (bez dostrajania wag BERTa).

Właściwie, to nie trzeba łączyć modelu BERT z własnymi warstwami, które mają rozwiązać zadanie docelowe. Wystarczy wygenerować reprezentacje przykładów tekstowych ze zbioru i zapisać do pliku. Na nich następnie trenuje się jedynie mały model klasyfikatora, co znacznie przyspiesza proces eksperymentowania oraz wytrenowania gotowego modelu, ale kosztem dokładności.

Metoda *fine-tuning* oferuje lepszą ogólną efektywność niż podejście *feature-based*, co autorzy ukazali zestawiając ze sobą różne wariacje tych podejść. W zadaniu CoNLL-2003 Named Entity Recognition (NER), *BERT_{BASE}* dotrenowany metodą *fine-tuning* osiąga wartość miary F1 na poziomie 96.4, podczas gdy model wytrenowany podejściem *feature-based* osiąga największą wartość 96.1, co stanowi różnicę wynoszącą 0.3 dla miary F1.

W fine-tuningu twórcy starają się stworzyć modele rozwiązuające między innymi zadanie Question Answering (QA), na zbiorze SQuAD v1.1 i SQuAD v2.0, które polega na tym, że mając jakiś paragraf z pytaniem w zdaniu A, chcemy poznać zakres tekstu w zdaniu B, który odpowiada na pytanie zawarte w zdaniu A. Także zadanie Named Entity Recognition (NER), którego celem jest odkrycie, które tokeny reprezentują encje nazwane (*Named Entities*) oraz jakiego są rodzaju. Encje nazwane to takie wyrazy zawarte w analizowanej frazie, które są imionami osób, nazwami organizacji i miejsc [75], np. fraza „[ORG U.N.] official [PER Ekeus] heads for [LOC Baghdad] .” – zawiera trzy encje nazwane: Ekeus jest osobą, U.N. jest organizacją, a Baghdad jest miejscem.

2.8.2 Model RoBERTa

RoBERTa [52] jest modelem opartym na architekturze BERTa. Poprawia jego efektywność dzięki: (1) trenowaniu modelu dłużej, z większą liczbą przykładów oraz większym rozmiarem batcha; (2) usunięciu zadania NSP; (3) trenowaniu dłuższych sekwencji; i (4) dynamicznej zmianie wzorca maskowania, który jest nakładany na dane ze zbioru treningowego.

Wykorzystywany jest *dynamic masking*, który polega na nakładaniu wzorca maskowania w czasie trenowania, podczas przekazywania przykładów do modelu, zamiast w czasie przygotowywania zbioru do treningu, jak ma to miejsce w przypadku BERTa (*static masking*).

Zadanie NSP zostało usunięte ze względu na zauważenie, że hipoteza postawiona w pracy na temat BERT, mówiąca że trenowanie również pod zadanie NSP umożliwia nauczenie się relacji pomiędzy zdaniami, jest błędna, ponieważ eksperyment porównujący oryginalny format wejściowy z alternatywnymi formatami, które testują pojedyncze zdania jak i całe ciągi następujących po sobie zdań z jednego lub wielu dokumentów, pokazuje, że **usunięcie zadania NSP nie zmienia lub delikatnie poprawia efektywność w docelowym zadaniu**. Autorzy sugerują, że być może twórcy BERTa usunęli uwzględnianie błędu NSP w funkcji straty, jednocześnie zachowując oryginalny format danych wejściowych z podziałem na zdanie A i B.

2.9 Miary

W celu oceny wydajności modelu oraz monitorowania postępów w procesie trenowania stosuje się różnego rodzaju miary. Podczas trenowania wykorzystywana jest zazwyczaj funkcja straty, która na podstawie różnicy pomiędzy wartościami rzeczywistymi a przewidywaniami modelu informuje o jakości dopasowania. Po zakończeniu trenowania oblicza się miary ewaluacyjne, które pozwalają porównać wydajność różnych modeli, architektur i podejść [7].

TABELA 2.1: Przykład tabeli pomyłek [7]

	spam (actual)	not_spam (actual)
spam (predicted)	23 (TP)	12 (FP)
not_spam (predicted)	1 (FN)	556 (TN)

2.9.1 Macierz pomyłek

W przypadku klasyfikacji konstruuje się tzw. macierz pomyłek, która jest kwadratową macierzą, gdzie liczba rzędów i kolumn odpowiada liczbie możliwych klas. Na osi rzędów są klasy, które model przewidział (przewidziane klasy - *predicted*), a na osi kolumn znajdują się klasy, które model powinien przewidzieć (faktyczne klasy - *actual*). Dla każdej pary faktycznej klasy oraz przewidzianej klasy zlicza się odpowiednie wystąpienia podczas ewaluacji modelu.

W literaturze spotyka się taką definicję lub podobne: dla każdego klasyfikatora $f : D \rightarrow C = \{1, \dots, n\}$ i skońzonego zbioru $S \subseteq D \times C$, niech $m^{f,S} \in \mathbb{N}_0^{n \times n}$ będzie macierzą pomyłek, gdzie $m_{ij}^{f,S} = |\{s \in S | f(s_1) = i \wedge s_2 = j\}|$ [30, 59].

W klasyfikacji binarnej problem upraszcza się i mamy macierz pomyłek o rozmiarze 2×2 . Przykład macierzy pomyłek dla binarnej klasyfikacji spamu można zaobserwować na Tabeli 2.1. Przykłady zaklasyfikowane przez model jako spam (*spam*), które faktycznie były spamem oznacza się jako *true positives* (TP) i jest ich 23. Przykłady zaklasyfikowane jako „nie spam” (*not_spam*), które w rzeczywistości są spamem, to *false negatives* (FN). Przykłady zaklasyfikowane jako spam, które w rzeczywistości nie są spamem to *false positives* (FP). Przykłady zaklasyfikowane jako „nie spam”, które w rzeczywistości nie są spamem, to *true negatives* (TN). Jak można się domyślić, przykłady TP i TN reprezentują poprawne przewidywania modelu, a FP i FN, to różne rodzaje pomyłek.

Gdy bada się TP, FN, FP i TN w klasyfikacji wieloklasowej, to wybiera się jedną z klas i to ją się traktuje jako *positive*, a wystąpienia innych klas w przykładach jako *negative*. Innymi słowy, problem zamienia się na wiele problemów binarnych - jeden dla każdej z klas. Ale to nie jest wymóg.

2.9.2 Precyzja i pełność

Macierz pomyłek jest wykorzystywana do obliczania skalarnych metryk, takich jak precyzja i pełność.

Precyzja (*precision*) mierzy stosunek liczby poprawnie wykrytych pozytywnych przypadków do wszystkich wykrytych pozytywnych przypadków. Niech P_i oznacza precyzję względem klasy i . Wtedy:

$$P_i = \frac{TP_i}{TP_i + FP_i} \quad \text{lub} \quad P_i = \frac{m_{ii}^{f,S}}{\sum_{x=1}^n m_{ix}^{f,S}}$$

Pełność (*recall*) mierzy stosunek poprawnie wykrytych pozytywnych przypadków do wszystkich faktycznych pozytywnych przypadków. Niech R_i oznacza pełność względem klasy i . Wtedy:

$$R_i = \frac{TP_i}{TP_i + FN_i} \quad \text{lub} \quad R_i = \frac{m_{ii}^{f,S}}{\sum_{x=1}^n m_{xi}^{f,S}}$$

$P_i \in [0, 1]$ i $R_i \in [0, 1]$. Wysoka precyzja często wiąże się z niższą pełnością i odwrotnie. Zjawisko to nazywa się *precision-recall trade-off*. W zależności od zastosowania można preferować jedną z

metryk — np. w filtrze spamu wysoka precyzja oznacza mniej fałszywych alarmów, a w diagnostyce medycznej często ważniejsza jest wysoka pełność, aby wykryć jak najwięcej przypadków.

2.9.3 F-score

Aby móc zawiązać w jedną skalarną metrykę zarówno precyzję i pełność, to stosuje się miarę *F-score*, która jest średnią harmoniczną precyzji i pełności. Średnia harmoniczna penalizuje duże rozbieżności między *precision* a *recall*, co jest kluczowe w tej metryce. Niech F_β oznacza wartość miary *F-score*. Wtedy miara ta, dla klasy i , jest zdefiniowana jako:

$$F_{\beta_i} = (1 + \beta^2) \cdot \frac{P_i \cdot R_i}{(\beta^2 \cdot P_i) + R_i}$$

$$F_{\beta_i} \in [0, 1]$$

Wartość β oznacza wagę, która mówi ile razy pełność jest ważniejsza niż precyzja. Zwykle stosuje się takie wartości β jak 2, która mówi, że pełność jest dwa razy ważniejsza niż precyzja, a w przypadku wartości 0.5 precyzja jest dwa razy ważniejsza niż pełność. W pracach dotyczących głębokiego uczenia stosuje się często metrykę *F-score* z wartością β równą 1, która jest szczególnym przypadkiem tej metryki:

$$F_{1_i} = \frac{2 \cdot P_i \cdot R_i}{P_i + R_i}$$

2.9.4 Dokładność

Dokładność jest globalnym stosunkiem liczby poprawnie zaklasyfikowanych przykładów do całkowitej liczby przykładów. Niech A oznacza dokładność modelu. Wtedy:

$$A = \frac{TP + TN}{TP + FP + FN + TN} \quad \text{lub} \quad A = \frac{\sum_{i=1}^n m_{ii}^{f,S}}{\sum_{i=1}^n \sum_{j=1}^n m_{ij}^{f,S}}$$

$$A \in [0, 1]$$

2.9.5 Rodzaje redukcji metryk

W problemach wieloklasowych na powyższych metrykach takich jak precyzja, pełność, dokładność i miarę F_β stosuje się redukcje [48, 73, 30, 59].

Rodzaje redukcji to:

Uśrednienie macro - definiuje się jako średnia arytmetyczna metryki na poszczególnych klasach. Można intuicyjnie założyć, że rodzaj redukcji traktuje ważność wszystkich klas tak samo, ponieważ zsumowane pomiary na wszystkich klasach są uśrednianie przez całkowitą liczbę klas. Przykład redukcji F1 macro, którą wykorzystujemy:

$$\mathcal{F}_1 = \frac{\sum_{i=1}^n F_{1_i}}{n} \tag{2.1}$$

Podobny wzór stosuje się do innych wymienionych metryk, z tą różnicą, że zamiast F1 wykorzystywana jest inna metryka M :

$$M_{macro} = \frac{\sum_{i=1}^n M_i}{n}$$

Istnieje również inna definicja metryki F1 macro, która definiuje ją jako średnią harmoniczną ze średnich arytmetycznych:

$$\mathbb{F}_1 = H(\bar{P}, \bar{R}) = \frac{2\bar{P}\bar{R}}{\bar{P} + \bar{R}} = 2 \frac{\left(\frac{1}{n} \sum_{i=1}^n P_i\right) \left(\frac{1}{n} \sum_{i=1}^n R_i\right)}{\frac{1}{n} \sum_{i=1}^n P_i + \frac{1}{n} \sum_{i=1}^n R_i}$$

Jednak my korzystamy ze wzoru na średnią arytmetyczną ze średnich harmonicznych zdefiniowaną we wzorze 2.1.

Uśrednienie micro - polega na uwzględnieniu wszystkich klas jednocześnie, bez brania pod uwagę możliwych różnic pomiędzy nimi. Stąd micro precyzję oblicza się następująco [30]:

$$P_{micro} = \frac{\sum_{i=1}^n TP_i}{\sum_{i=1}^n TP_i + FP_i}$$

Micro pełność wynosi dokładnie tyle samo co micro precyzja, ze względu na identyczny licznik, który zlicza prawdziwe pozytywy z różnych klas, i identyczny mianownik.

$$R_{micro} = \frac{\sum_{i=1}^n TP_i}{\sum_{i=1}^n TP_i + FN_i}$$

Uśrednianie ważone jest obliczane przy pomocy wag, które są określane jako stosunek udziału przykładów z danej klasy do całego zbioru. W ten sposób możliwe jest bardziej sprawiedliwe oszacowanie wkładu poszczególnych klas w ostateczną wartość miary, co może być przydatne w przypadku niezbalsowanego zbioru. Przykładowo, precyzję można obliczyć w ten sposób z uśrednianiem ważonym:

$$P_{weighted} = \sum_{k=1}^n w_k P_k \quad \text{gdzie} \quad w_k = \frac{N_k}{N}$$

gdzie N_k , to liczba faktycznych przykładów klasy k (czyli suma kolumny k w macierzy pomyłek):

$$N_k = \sum_{l=1}^n m_{lk}^{f,S}$$

a N , to całkowita liczba przykładów w zbiorze:

$$N = \sum_{i=1}^n \sum_{j=1}^n m_{ij}^{f,S}$$

Czyli pełny wzór to:

$$P_{weighted} = \frac{1}{N} \sum_{k=1}^n N_k \cdot P_k = \frac{1}{\sum_{i=1}^n \sum_{j=1}^n m_{ij}^{f,S}} \sum_{k=1}^n \left(\sum_{l=1}^n m_{lk}^{f,S} \right) P_k$$

2.9.6 Entropia

Entropia w teorii informacji mówi **ile średnio bitów jest potrzebnych, aby zakodować zmienną losową X** , biorąc pod uwagę jej rozkład prawdopodobieństwa $P(X)$ [5]. Zakładając, że X jest zmienną losową przyjmującą jedną z wartości x_1, x_2, \dots, x_n , z prawdopodobieństwem p_1, p_2, \dots, p_n , to entropia jest zdefiniowana następująco:

$$H(X) = - \sum_{i=1}^n p_i \cdot \log_2 p_i$$

Wyrażenie $-\log_2 p_i$ reprezentuje liczbę potrzebnych bitów do zakodowania pojedynczego możliwego zdarzenia zmiennej losowej X . Intuicyjnie można zauważać, że wzór ten w pewien sposób przypomina wzór na wartość oczekiwana zmiennej losowej X , gdzie p_i jest wagą w średniej, a $-\log_2 p_i$ reprezentowałby miarę „zdziwienia”. Wysoko prawdopodobne zdarzenia praktycznie nie wymagają transmisji informacji, w przeciwieństwie do mało prawdopodobnych zdarzeń, więc można to interpretować tak, że im mniej prawdopodobne zdarzenie, tym większe „zdziwienie” powinno generować.

Wysoko skupione rozkłady prawdopodobieństw mają niską entropię, a rozkłady równomierne wysoką. Należy jednocześnie zauważać, że $\lim_{p_i \rightarrow 0} p_i \log_2 p_i = 0$, więc gdy $p_i = 0$, to dla umożliwienia obliczeń można założyć, że wtedy $\log_2 p_i = 0$. Do obliczeń częściej używa się podstawy e w logarytmie. My do analiz stosujemy logarytm o podstawie 2, ponieważ niesie ze sobą dodatkową informację na temat potrzebnych bitów, co jest bardziej intuicyjne niż użycie jednostek *nat*.

Entropia krzyżowa to rozszerzenie idei entropii. Polega na obliczeniu entropii zmiennej losowej X , o prawdopodobieństwach zdarzeń p_1, p_2, \dots, p_n , przy zastosowaniu rozkładu innej zmiennej losowej Y , o prawdopodobieństwach q_1, q_2, \dots, q_n .

$$H(X, Y) = - \sum_{i=1}^n p_i \log_2 q_i$$

W uczeniu maszynowym prawdopodobieństwo klas przewidziane przez model \vec{y} , gdzie $q_i = \vec{y}_i$, jest porównywane w prawdziwymi wartościami prawdopodobieństw przykładów \vec{y} , gdzie $p_i = \vec{y}_i$, przy pomocy entropii krzyżowej. Z tej idei korzysta funkcja straty entropii krzyżowej.

W PyTorchu [25] klasa CrossEntropyLoss implementuje ten wzór, ale w nieco zmienionej formie. Wynikiem modelu są logity, które przetworzone przy pomocy funkcji softmax, zamieniają się na rozkład prawdopodobieństw. W klasyfikacji klasowej istotna jest jedynie jedna klasa wynikowa, więc z wyniku działania modelu bierze się indeks maksymalnego argumentu, który koncepcyjnie reprezentuje pewien wektor one-hot \vec{z} . Wtedy równanie entropii krzyżowej upraszcza się do $H(X, Y) = -\log \text{softmax}(\vec{z})$. Wzór wykorzystywany przez klasę z PyTorchu to $l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore_index}\}$, gdzie $\frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})}$ reprezentuje funkcję softmax obliczoną na wyjściu x_n modelu, gdzie:

- l_n - wartość funkcja straty dla pojedynczego przykładu n . Ostatecznie klasa robi redukcję na wszystkich l_n i powstaje $\ell(x, y)$ - czyli oblicza średnią arytmetyczną z l_n lub je sumuje.
- y_n - indeks docelowej klasy w tym przykładzie.
- c - indeks każdej z klas, co powoduje, że $x_{n,c}$ jest wyjściem modelu dla przykładu n dla każdej z klas $c \in \{1, 2, \dots, C\}$.
- w_{y_n} - wagi prowadzące do wyjścia o indeksie y_n .

Parametr *ignore_index* reprezentuje indeks klasy, którą wzór ma ignorować, np. wartość 255, która mogłaby reprezentować nieistotny piksel w problemie segmentacji obrazów.

2.10 Trudności związane z klasyfikacją prawdziwości treści

2.10.1 Złożoność formatów wejściowych

Istnieje szereg problemów, które utrudniają zadanie klasyfikacji prawdziwości treści. Występujące powszechnie treści wykorzystują zróżnicowane formaty, takie jak: tekst, obraz, wideo, dźwięk i ich różne multimodalne połączenia [79].

2.10.2 Istotność kontekstu informacji

Wiele stwierdzeń ma złożony charakter i posiada wiele niuansów, które są trudne do zrozumienia przez maszynę. Złożoność i gra językowa wpływa na możliwość analizy treści, co jest problemem już w oderwaniu od postawionego zadania, ponieważ różne zabiegi takie jak kluczenie, czy użycie sarkazmu w fałszywych treściach, jest już samo w sobie trudnym zadaniem klasyfikacyjnym. Jednak warto zauważyć, że może ono być przydatne w analizie prawdziwości stwierdzeń.

Inaczej niż bezpośrednia klasyfikacja tekstu, klasyfikacja prawdziwości treści wymaga zrozumienia kontekstu, zamiaru i umiejętności odniesienia się do informacji z różnych źródeł. Kontekst, np. czasowy lub tematyczny, może mieć duży wpływ na możliwość właściwego oszacowania prawdziwości treści. Nagłówki informacji są często krótkie. Przykładowo nagłówek „Stocks lose ground amid inflation concerns, trade war worries” [10], zobaczy w sieci społecznościowej, mógłby wprowadzić kogoś w błąd, gdyby został udostępniony w sieci społecznościowej z dużym odstępem czasowym od daty jego pierwotnego wydania.

2.10.3 Rodzaje fałszywych informacji

Treści mogą przyjąć wiele postaci, od całkowicie zmyślonych do częściowo prawdziwych informacji, ale w fałszywym kontekście. Cel treści może być np. nieświadomym wprowadzaniem w błąd (mylną informacją) lub celowym oszustwem (dezinformacją) [11].

2.10.4 Halucynacja LLM

Istotnym problemem w generacji tekstu przy pomocy metod uczenia maszynowego jest zjawisko „halucynacji”, które objawia się tym, że generatywny model LLM generuje prawdopodobne, ale faktycznie niepoprawne lub bezsensowne informacje [80]. Niestety, jeśli zadaniem jest klasyfikacja treści z użyciem automatycznie wygenerowanych treści, to ten problem może negatywnie wpływać na efektywność zaimplementowanego rozwiązania.

2.11 Technologie

Wykorzystane technologie to:

- **CUDA** - platforma obliczeniowa rozwijana przez firmę Nvidia do przetwarzania równoległego, która umożliwia wykorzystanie procesorów graficznych do wykonywania obliczeń ogólnego przeznaczenia, co znaczco zwiększa wydajność w zadaniach wymagających dużej mocy obliczeniowej.
- **Jupyter Notebook** - interaktywne środowisko programistyczne wspierające analizę danych, uczenie maszynowe oraz wizualizację wyników. Umożliwia tworzenie i uruchamianie dokumentów zawierających kod, wykresy i tekst formatowany w języku Markdown.
- **DVC** - narzędzie wspierające zarządzanie danymi i modelami w projektach uczenia maszynowego. Umożliwia wersjonowanie zbiorów danych i śledzenie eksperymentów.
- **MLflow** - narzędzie, które umożliwia śledzenie eksperymentów, uruchomień w eksperymentach, hiperparametrów uruchomień, metryk trenowania, a także pozwala na ewaluację wytrenowanych modeli, deployment, zapis modeli i innych artefaktów z procesu trenowania, porównywanie eksperymentów oraz posiada integracje z różnymi popularnymi bibliotekami i platformami jak np. HuggingFace, TensorFlow, PyTorch, scikit-learn i inne.

- **Pandas** - biblioteka języka Python przeznaczona do analizy i przetwarzania danych w formie tabelarycznej.
- **SQLite** - system zarządzania relacyjnymi bazami danych, który nie wymaga instalacji osobnego serwera.
- **Paramiko** - biblioteka języka Python umożliwiająca bezpieczną komunikację z użyciem protokołu SSHv2.
- **TorchMetrics** - biblioteka języka Python, która umożliwia łatwe śledzenie metryk modeli uczenia maszynowego.
- **LanguageTool** - otwartoźródłowe narzędzie służące do sprawdzania poprawności gramatycznej tekstów w różnych językach. Jest używany w OpenOffice. Udostępnia nakładkę w postaci pakietu `language-tool-python` w Pythonie.
- **scikit-learn** - biblioteka języka Python, która oferuje szeroki zestaw narzędzi do klasyfikacji, regresji, grupowania, redukcji wymiarowości oraz przetwarzania danych, wraz z prostym interfejsem do trenowania i oceny modeli.
- **Gradio** - biblioteka języka Python, która służy do tworzenia aplikacji webowych umożliwiających prezentację i testowanie modeli uczenia maszynowego.

Rozdział 3

Wybór zbioru danych

Z racji na dużą dostępność anglojęzycznych zbiorów danych rozważaliśmy kilka różnych zbiorów danych: Fakeddit [56], FEVER [74], NELA-GT-2020 [31], COVID-19 Fake News Dataset [60], LIAR [78], LIAR PLUS [3].

Fakeddit to multimodalny zbiór danych, składający się z ponad 1 miliona przykładów. Został stworzony na podstawie wielu subredditów z Redditu. Oznaczenie klasy przykładów odbywa się na zasadzie tematycznej przynależności subredditu, z którego pobrano przykłady. Proces pozyskiwania danych zawiera w sobie ręczną weryfikację poprawności na malej próbce danych. Przykłady składają się z pojedynczej 2-stopniowej, 3-stopniowej i 6-stopniowej cechy celu klasyfikacji, komentarza, metadanych, treści i obrazu. Poza tym, że praca Fakeddit wprowadza nowy zbiór danych, to badacze jednocześnie pokazują, że zastosowanie multimodalnego zbioru danych poprawia jakość klasyfikacji.

FEVER to zbiór danych, stworzony z twierdzeń zaczerpniętych z Wikipedii, posiadający 185,445 przykładów. Każdy przykład składa się z twierdzenia, listy tekstowych dowodów i werdyktu. Werdykt jest cechą, zaklasyfikowaną przez annotatorów, reprezentującą jedną z klas: SUPPORTED, REFUTED i NOTENOUGHINFO. Dla pierwszych dwóch z klas anotatorzy dodali również listę dowodów wspierających ich ocenę prawdziwości twierdzenia.

NELA-GT-2020 jest zbiorem danych zbudowanym z 1.8 miliona artykułów prasowych zebraanych z 519 źródeł pomiędzy 1 stycznia 2020 roku aż do 31 grudnia 2020 roku. NELA-GT-2020 jest następcą zbioru NELA-GT-2019 i rozszerza go o dodatkowe informacje w postaci treści postów na platformie X (wcześniej nazywala się Twitter) zamieszczonych wewnątrz pozyskanych artykułów. Każdy przykład posiada etykietę, wyodrębnioną przy pomocy serwisu Media Bias/Fact Check¹, która może przyjąć wartość z pomiędzy kilku klas: UNRELIABLE, MIXED lub RELIABLE. Zbiór danych można uzyskać w dwóch formatach: SQLite lub zbiór plików JSON, gdzie każdy plik reprezentuje źródło z którego artykuły zostały uzyskane, a wewnątrz pliku znajdują się przykłady z tego źródła. Baza danych SQLite zawiera dwie tabele **newsdata**, która przechowuje treści artykułów i **tweet**, która przechowuje treści postów na platformie X przypisanych do danego artykułu. Dane w zbiorze były zbierane na bieżąco przez skrypt odczytujący zawartości artykułów i pomiędzy 25 marca 2020 roku do 8 kwietnia 2020 roku nastąpiła awaria, której wynikiem są braki danych w zbiorze w tym okresie. Warto zaznaczyć, że te braki stanowią 0.8% zawartości całego zbioru. Badacze ukazują kilka możliwych zastosowań tego zbioru danych:

- Badanie pokrycia tematu COVID-19 przez media prasowe.

¹<https://mediabiasfactcheck.com>

- Badanie narracji i pokrycia tematu w tle wyborów prezydenckich z 2020 roku w Stanach Zjednoczonych.
- Badanie relacji pomiędzy tweetami a różnymi artykułami. Jeden tweet może być częścią wielu artykułów i z tego powodu badacze zauważają, że stworzenie strukturalnej sieci powiązań pomiędzy tweetami a artykułami może pomóc w identyfikacji sygnałów wskazujących na niezawodność źródła.

COVID-19 Fake News Dataset to zbiór danych skupiony tematycznie na informacjach na temat pandemii COVID-19 uzyskanych z postów na mediach społecznościowych i artykułów. Składa się z 10,700, ręcznie zaklasyfikowanych, przykładów prawdziwych i fałszywych informacji na temat pandemii COVID-19. Rekordy zbioru są oznaczone binarnie na klasy REAL, FAKE. Każdy element zbioru składa się z kolumn *Label*, *Source* i *Text*, gdzie:

- *Label* to klasa oznaczająca prawdziwość informacji (REAL, FAKE).
- *Source* to źródło pochodzenia, np. Facebook, Twitter (WHO), Fact checking.
- *Text* to tekstowa treść informacji.

Pole *Text* może zawierać odnośniki do stron internetowych, a także nazwy użytkowników poprzedzone znakiem '@', które oznaczają odniesienie do czyjegoś profilu na platformie X (kiedyś Twitter). Zbiór COVID-19 Fake News Dataset jest podzielony na trzy podzbiory: treningowy (60%), testowy (20%) i walidacyjny (20%). 52.34% elementów w zbiorze to informacje oznaczone jako REAL (prawda), a 47.66% to FAKE (fałsz). Twórcy zbioru informują, że utrzymują równomierną dystrybucję klas na wszystkich zbiorach.

Podczas sprawdzenia zbioru LIAR odkryliśmy źródło danych, na którym opiera się wiele innych zbiorów związanych z fact-checkingiem. Jest nim amerykańska strona internetowa POLITI-FACT.COM. Powstała w 2007 roku z inicjatywy redakcji Tampa Bay Times (kiedyś St. Petersburg Times), a obecnie jest zarządzana przez instytucję non-profit Poynter Institute for Media Studies. Strona zajmuje się oceną prawdziwości poprzez przypisywanie różnym informacjom etykiety w formie tzw. Truth-O-Meter, który jest skalą sześciu klas prawdziwości informacji:

- „Pants on Fire” - Oświadczenie nie jest dokładne i jest absurdalne.
- „False” - Oświadczenie nie jest poprawne.
- „Mostly False” - Oświadczenie zawiera element prawdy, ale ignoruje krytyczne fakty, które sprawiłyby inne wrażenie.
- „Half True” - Oświadczenie jest częściowo dokładne, ale pomija ważne szczegóły lub usuwa rzeczy z kontekstu.
- „Mostly True” - Oświadczenie jest dokładne, ale wymaga wyjaśnienia lub dodatkowych informacji.
- „True” - Oświadczenie jest dokładne i nie brakuje w nim niczego znaczącego.

Serwis zajmuje się zarówno bieżącymi wydarzeniami jak i starszymi wypowiedziami, które zyskują popularność. Weryfikuje zarówno polityków różnych partii, jak i organizacje, media oraz popularne łańcuszki internetowe. W Polsce istnieje odpowiednik tego serwisu, którym jest demagog.org.pl.

LIAR to zbiór 12.8 tysięcy ręcznie oznaczonych przykładów, przypisanych do wielu tematów, uzyskanych z serwisu POLITIFACT.COM. Nadaje się do fact-checkingu oraz detekcji fake-newsów. Został utworzony w wyniku dużego nagromadzenia różnego rodzaju fake-newsów podczas wyborów prezydenckich na czterdziestego piątego prezydenta Stanów Zjednoczonych Ameryki. Przykłady w zbiorze skupiają się na różnych krótkich wypowiedziach, w tym na wypowiedziach polityków, opublikowanych na różnych platformach i sprawdzonych przez serwis POLITIFACT.COM. Każdy przykład składa się z:

- Przyporządkowanej klasy prawdziwości, która może przyjąć jedną z 6 wartości od całkowicie fałszywej wypowiedzi do prawdziwej wypowiedzi.
- Tematu wypowiedzi.
- Kontekstu wypowiedzi.
- Kolumny oznaczającą autora albo źródło wypowiedzi, np. Barack Obama.
- Stan, z którego mówca pochodzi.
- Partię, do której mówca aktualnie należy (na czas powstania zbioru).
- Dotychczasowa historia wypowiedzi w postaci wektora liczby przyporządkowanych do każdej z klas wypowiedzi danego mówcy (np. ile razy ktoś skłamał, ile razy ktoś powiedział półprawdę, itd...).

Zbiór jest podzielony na 3 podzbiory: treningowy (10,269 przykładów), testowy (1,283 przykłady) i walidacyjny (1,284 przykłady).

LIAR PLUS jest rozszerzeniem zbioru LIAR o, automatycznie wyodrębnioną z oryginalnych artykułów na POLITIFACT.COM, kolumnę *justification*, która opisuje przyczynę, dla której dany przykład został zaklasyfikowany do danej klasy decyzyjnej. Autorzy zbioru pokazują, że użycie tej informacji w modelu poprawia wartość miary F1 modelu zarówno dla klasyfikacji binarnej, gdzie klasy *pants on fire*, *false* i *mostly false* zamieniane są w jedną FALSE, a *true*, *mostly true* i *half true* w TRUE, jak i dla klasyfikacji sześcioklasowej na obu zbiorach testowym i walidacyjnym.

Po rozpoczęciu pracy nad projektem, odkryliśmy, że w lipcu 2024 roku został upubliczniony zbiór **LIAR2**, którego autorzy poprawili niektóre niedociągnięcia i braki w zbiorze LIAR i LIAR PLUS. Rozszerzyli go także prawie dwukrotnie z 12.8 tysiąca do 23 tysięcy przykładów. Autorzy zaproponowali także nowy model FDHN (Fuzzy Deep Hybrid Network), który integruje logikę rozmytą z głębokim uczeniem oraz pokazali, że zastosowanie tego modelu poprawia dokładność wywodzenia modelu i wartość metryki F1 na podzbiorze testowym i walidacyjnym [79].

LIAR2 zawiera większą liczbę przykładów, poprawki w istniejących rekordach oraz dodatkowe informacje niedostępne w poprzednich wersjach, takie jak liczba prawdziwych wypowiedzi mówcy. Niemniej jednak, w kontekście niniejszej pracy zdecydowano się na użycie zbioru LIAR PLUS ze względu na jego powszechnie wykorzystanie w literaturze oraz zgodność z oryginalną strukturą zbioru LIAR, co ułatwia porównywalność wyników z wcześniejszymi badaniami.

Rozdział 4

Eksploracyjna analiza danych

Zanim przeprowadziliśmy właściwą analizę zbioru, zaimplementowaliśmy po kilka skryptów rozszerzających oryginalny zbiór danych o dodatkowe kolumny, mając nadzieję, że pozwoli nam to zauważycie więcej zależności w zbiorze. Wykorzystaliśmy narzędzie **DVC** do stworzenia sekwencji kroków, dzięki której zrealizowaliśmy wiele zależnych od siebie skryptów, gdzie każdy kolejny dodaje nową kolumnę do wyniku poprzedniego. Podczas generowania dodatkowych danych zauważaliśmy niepoprawnie wczytane przykłady ze zbioru. Dopiero gdy wygenerowaliśmy potrzebne kolumny, wykonaliśmy docelową analizę danych. Analizę danych przeprowadziliśmy przy pomocy paczki **pandas** w języku programowania **Python** z użyciem środowiska **Jupyter Notebook**.

4.1 Opis zbioru danych

Zbiór LIAR PLUS składa się 12,836 przykładów. Jest podzielony na trzy podzbiory: treningowy - **train2.tsv**, testowy - **test2.tsv** i walidacyjny - **val2.tsv**. Zbiór treningowy składa się z 10,269 przykładów, zbiór testowy składa się z 1,283 przykładów, a zbiór walidacyjny składa się z 1,284 przykładów. Wszystkie podzbiory zbioru są zapisane w formacie TSV.

Każdy przykład posiada następujące cechy:

- Kolumna 1 - *id* - Identyfikator rekordu.
- Kolumna 2 - *json_id* - Identyfikator wypowiedzi.
- Kolumna 3 - *label* - Etykieta opisująca stopień prawdziwości wypowiedzi. Możliwe etykiety to (w kolejności od najbardziej fałszywej do najbardziej prawdziwej): PANTS-FIRE, FALSE, BARELY-TRUE, HALF-TRUE, MOSTLY-TRUE, TRUE.
- Kolumna 4 - *statement* - Klasyfikowana wypowiedź (lub temat poruszany w artykule na PolitiFact).
- Kolumna 5 - *subject* - Temat lub lista tematów, których dotyczy klasyfikowana wypowiedź.
- Kolumna 6 - *speaker* - Autor wypowiedzi.
- Kolumna 7 - *job_title* - Praca lub rola, którą wykonuje osoba, która wypowiedziała daną wypowiedź.
- Kolumna 8 - *state* - Nazwa stanu z którego jest osoba, która wypowiedziała daną wypowiedź (również może to być nazwa państwa).
- Kolumna 9 - *party_affiliation* - Przynależność partyjna autora wypowiedzi.

- Kolumny 10-14 - Całkowita liczba wypowiedzi danej osoby zaklasyfikowanych do konkretnej etykiety, zaliczając też obecną. W kolejności:
 - 10 - *barely_true_counts*: liczba wypowiedzi zaklasyfikowanych jako BARELY-TRUE.
 - 11 - *false_counts*: liczba wypowiedzi zaklasyfikowanych jako FALSE.
 - 12 - *half_true_counts*: liczba wypowiedzi zaklasyfikowanych jako HALF-TRUE.
 - 13 - *mostly_true_counts*: liczba wypowiedzi zaklasyfikowanych jako MOSTLY-TRUE.
 - 14 - *pants_on_fire_counts*: liczba wypowiedzi zaklasyfikowanych jako PANTS-FIRE.
- Kolumna 15 - *context* - Informacje o tym, gdzie lub w jakich okolicznościach dana wypowiedź została wypowiedziana.
- Kolumna 16 - *justification* - Wytlumaczenie zaklasyfikowania danej wypowiedzi do danej etykiety, pobierane z artykułów fact-checkingowych. Dokładna metoda stworzenia tej kolumny jest opisana w [3].

Poza wyżej wymienionymi kolumnami znajdującymi się w zbiorze, rozszerzyliśmy zbiór o dodatkowe kolumny metadanych. Niektóre z nich istnieją tylko na potrzeby analizy i nie są zapisane w plikach zbiorów, a inne zapisaliśmy w plikach zbiorów, ponieważ ponowne ich wyliczenie zajmuje dużo czasu.

Dodane kolumny i ich opis:

- Kolumna 17 - *sentiment* - Informacja o sentymencie zawartym w treści kolumny *statement*.
- Kolumna 18 - *question* - Informacja o tym, czy treść kolumny *statement* jest pytaniem czy zdaniem twierdzącym.
- Kolumna 19 - *grammar_errors* - Liczba błędów gramatycznych w treści kolumny *statement*.
- Kolumna 20 - *ratio_of_capital_letters* - Procentowa liczba dużych znaków w treści kolumny *statement*.
- Kolumna 21 - *curse* - Informacja o tym, czy treść kolumny *statement* jest wulgarna.
- Kolumna 22 - *emotion* - Informacja o tym, jakie emocje w czytającym może wywołać treść kolumny *statement*.
- Kolumna 23 - *gibberish* - Informacja o tym jak bardzo treść kolumny *statement* jest składnym lub belkotliwym tekstem.
- Kolumna 24 - *offensiveness* - Informacja o tym czy treść kolumny *statement* jest obraźliwa.
- Kolumna 25 - *political_bias* - Informacja o tym jakiej opcji politycznej jest treść kolumny *statement*.
- Kolumna 26 - *articles* - Wygenerowany artykuł.

Dodatkowe kolumny przeznaczone tylko do analizy:

- Kolumna 27 - *statement_len* - Ilość znaków w treści kolumny *statement*.
- Kolumna 28 - *justification_len* - Ilość znaków w treści kolumny *justification*.
- Kolumna 29 - *statement_wc* - Ilość słów w treści kolumny *statement*.

- Kolumna 30 - *justification_wc* - Ilość słów w treści kolumny *justification*.
- Kolumna 31 - *bin_label* - Klasa do jakiej przyporządkowany jest przykład. Może przyjąć wartość TRUE albo FALSE. Jest wyznaczana na podstawie kolumny *label*, gdzie klasy PANTS-FIRE, FALSE i BARELY-TRUE są przypisane do jednej klasy FALSE, a klasy HALF-TRUE, MOSTLY-TRUE i TRUE są przyporządkowane do klasy TRUE.
- Kolumna 32 - *label_score* - Wartość liczbową wyznaczoną na postawie kolumny *label*, gdzie: PANTS-FIRE jest przyporządkowane wartości -3; FALSE jest przyporządkowane wartości -2; BARELY-TRUE jest przyporządkowane wartości -1; HALF-TRUE jest przyporządkowane wartości 1; MOSTLY-TRUE jest przyporządkowane wartości 2; TRUE jest przyporządkowany wartości 3.
- Kolumna 33 - *punctuations* - Liczba znaków interpunkcyjnych na pierwsze 100 znaków w treści kolumny *statement*.
- Kolumna 34 - *articles_wc* - Liczba słów w treści kolumny *articles*.

4.2 Generacja dodatkowych metadanych

Igor Santarek zaimplementował skrypty rozszerzające zbiór danych o informację o: sentymencie zawartym w treści; zliczanie liczby błędów gramatycznych w treści; informację o tym, czy treść jest pytaniem; liczbę kropek na 100 znaków w treści; liczbę słów w treści; liczbę słów w uzasadnieniu przypisania do klasy.

Mateusz Sztefek zaimplementował skrypty rozszerzające zbiór danych o: stronniczość polityczną wynikającą z treści; informację o tym, czy treść jest ofensywna; informację o tym, czy treść jest belkotem; informację o przeważającej emocji zawartej w treści; informację o tym, czy treść jest bardziej przekleństwem czy zwykłym tekstem.

Do wygenerowania informacji na temat sentymentu wykorzystaliśmy model z platformy Hugging Face – `cardiffnlp/twitter-roberta-base-sentiment`. To model oparty na architekturze RoBERTa, wytrenowany na 58 milionach postów na platformie X z dotrenowaniem nakierowanym na klasyfikację sentymentu tekstu napisanego w języku angielskim. Wynikiem klasyfikacji może być jedna z 3 klas: 0 - NEGATIVE, 1 - NEUTRAL lub 2 - POSITIVE.

Zliczenie liczby błędów gramatycznych w treści zostało przeprowadzone przy użyciu pakietu `language-tool-python`. Użycie pakietu pozwala wykrycie różnego rodzaju błędów w tekście i dokładną analizę. Użyliśmy go do wyliczenia liczby błędów gramatycznych i zbadania ich rodzajów w treści kolumny *statement*. W zbiorze `train2.tsv`, 5 najbardziej popularnych błędów gramatycznych pojawiających się w treści kolumny *statement* to: możliwy błąd w pisowni; niepoprawna pisownia form ściągniętych w języku angielskim; użycie białych znaków przed przecinkiem lub przed albo za nawiasem; potencjalny brak apostrofów; przecinek między niezależnymi klauzulami. Dokładną liczbę błędów opisuje Tabela 4.2. Listę i opis możliwych do wykrycia błędów gramatycznych przez LanguageTool można znaleźć pod adresem <https://community.languagetool.org/rule/list?lang=en-US>.

Informacja o trybie pytającym lub twierdzącym treści w kolumnie *statement* jest generowana przy użyciu modelu `mrsinghania/asr-question-detection` z platformy Hugging Face. Nie jest jasno zdefiniowane na jakiej architekturze oparty jest ten model, a także skąd dokładnie pochodzą przykłady, na których został wytrenowany, ale można przeczytać, że „klasyfikator został wyszkolony na ponad 7 tysiącach próbek pochodzących z wypowiadanych podczas wywiadów rozmów” [55]. Możliwym wynikiem działania modelu jest: 0 - STATEMENT, 1 - QUESTION. Przy dodawaniu

kolumny *question* do zbioru zapisywaliśmy klasę STATEMENT jako klasę NOT_QUESTION, a klasę QUESTION jako klasę QUESTION.

Liczę kropki na 100 znaków dodaliśmy w analizie przy pomocy funkcji dostępnych w pakiecie **pandas**. Tak samo zrobiliśmy z liczbą słów w kolumnie *statement* oraz w kolumnie *justification*.

Informacja o tym, czy treść kolumny *statement* jest wulgarna, jest generowana przy pomocy modelu **djsull/curse_classification** dostępnego na platformie Hugging Face. Nie ma podanych informacji o jego architekturze oraz zbiorze danych, na którym został wytrenowany. Wynik klasyfikacji może być następujący: CURSE lub NON-CURSE.

Dane o emocjach zostały wygenerowane przy pomocy modelu z platformy Hugging Face – **j-hartmann/emotion-english-distilroberta-base** [34]. Model ten jest oparty na architekturze RoBERTa z dotrenowaniem nakierowanym na klasyfikowanie emocji zawartych w tekście. Model został wytrenowany na zbalansowanym podzbiorze danych (2,811 obserwacji dla każdej emocji). 80% zbioru było wykorzystywane do szkolenia, a 20% do oceny. Dokładność oceny wynosi 66% w porównaniu do wartości bazowej 1/7 (14%). Możliwe wyniki klasyfikacji to: ANGER, DISGUST, FEAR, JOY, NEUTRAL, SADNESS lub SURPRISE.

Informację o poziomie bełkotliwości tekstu zostały wygenerowane przy pomocy modelu dostępnego na platformie Hugging Face – **madhurjindal/autonlp-Gibberish-Detector-492513457** [41]. Możliwe wyniki klasyfikacji to:

- NOISE - przykład: „dfdfer fgerfow2e0d qsqskdsd djksdnfkff swq”.
- WORD SALAD - przykład: „22 madhur old punjab pickle chennai”.
- MILD GIBBERISH - przykład: „Madhur study in a teacher”.
- CLEAN - przykład: „I love this website”.

Do wygenerowania informacji o tym, czy treść kolumny *statement* jest obraźliwa, został wykorzystany model z platformy Hugging Face – **cardiffnlp/twitter-roberta-base-offensive**. Model ten jest oparty na architekturze RoBERTa z dotrenowaniem nastawionym na identyfikację obraźliwych treści. Do wytrenowania wykorzystano około 58 milionów wpisów z platformy X. Możliwe wyniki klasyfikacji to: NOT-OFFENSIVE lub OFFENSIVE.

Informację o tym, jakiej opcji politycznej jest treść kolumny *statement*, zostały wygenerowane przy pomocy modelu **bucketresearch/politicalBiasBERT**, dostępnego na platformie Hugging Face [4]. Model oparty jest na architekturze BERT z dotrenowaniem nastawionym na wykrywanie przynależności politycznych tekstów, które są nacechowane politycznie. Dostępne podziały polityczne to: prawica (RIGHT), lewica (LEFT) i centrum (CENTER).

4.3 Oczyszczenie i analiza zbioru danych

4.3.1 Oczyszczenie zbioru

W trakcie generowania dodatkowych kolumn zauważliśmy, że w plikach **test2.tsv** i **train2.tsv** są uszkodzone rekordy i je ręcznie poprawiliśmy. W pliku **test2.tsv** zauważliśmy, że od 704 linii do 726 i od 946 do 956 linii, wartości kolumny *context* mają na końcu ciągu znak nowej linii. W pliku **train2.tsv** zauważliśmy taki sam problem w liniach od 1281 do 1305, 2167 do 2170, 6147 do 6161, 6215 do 6217, 7594 do 7610, 9435 do 9436. Dodatkowo, w tym samym pliku, na liniach 1281, 2155, 7572 i 9405 zauważliśmy błąd polegający na tym, że początek kolumny *statement* był poprawnie oznaczony znakiem cudzysłowia " , a koniec tej kolumny był oznaczony dwoma znakami pojedynczego cudzysłowia ' ' przez co rekord nie był poprawnie odczytywany i

TABELA 4.1: Cechy statystyczne liczbowych kolumn we wszystkich zbiorach

Kolumna	mean	std	min	25%	50%	75%	max
barely_true_counts	11.596	18.997	0.000	0.000	2.000	12.000	70.000
false_counts	13.370	24.151	0.000	0.000	2.000	15.000	114.000
half_true_counts	17.219	35.911	0.000	0.000	3.000	13.000	160.000
mostly_true_counts	16.527	36.226	0.000	0.000	3.000	12.000	163.000
pants_on_fire_counts	6.246	16.163	0.000	0.000	1.000	5.000	105.000
grammar_errors	0.382	0.671	0.000	0.000	0.000	1.000	7.000
ratio_of_capital_letters	3.686	2.684	0.000	1.890	3.170	4.760	82.140
statement_len	106.286	45.013	11.000	73.000	99.000	133.000	395.000
justification_len	426.692	309.836	3.000	272.000	395.000	537.000	9394.000
statement_wc	17.915	7.748	2.000	12.000	17.000	22.000	66.000
justification_wc	72.078	64.631	1.000	46.000	67.000	90.000	3030.000
label_score	0.271	2.008	-3.000	-2.000	1.000	2.000	3.000
punctuations	1.938	1.536	0.000	1.000	2.000	3.000	16.000

wtedy funkcja wczytująca plik TSV, wczytywała ten rekord i następny – jako jeden rekord, aż do natrafienia znaku zamykającego kolumnę ", np. "The vast majority of the money I got was from small donors all across the country." zamiast "The vast majority of the money I got was from small donors all across the country.". "

Pomimo powyższych poprawek, to później na etapie analizy wyników w rozdziale 7.3 i tak zauważaliśmy, że niektóre przykłady mają przypisane niepoprawne wartości uzasadnienia. Niektóre przykłady mają przypisane uzasadnienie z innych artykułów niż powinny. Sprawdziliśmy, czy nie popełniliśmy błędu przy poprawianiu błędów formatu CSV w zbiorach, ale zauważaliśmy, że w oryginalnych plikach zbioru LIAR PLUS ten problem również występuje, np. przykład o *id* 5695 (linia 5710) z pliku **train2.tsv** posiada treść *justification*, która powinna być przypisana do przykładu o *id* 5682 (linia 5697) [39].

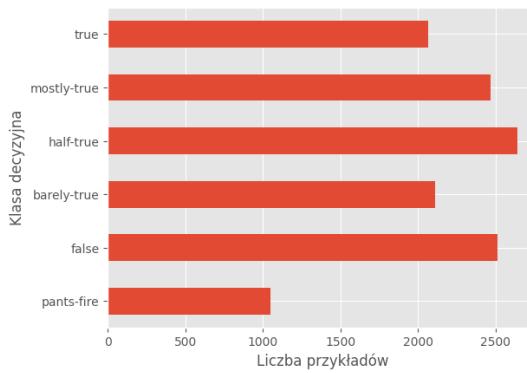
4.3.2 Analiza zbioru

Analiza została przeprowadzona na całym zbiorze. Statystyki kolumn liczbowych zostały przedstawione w Tabeli 4.1. W analizowanym zbiorze danych średnia długość zawartości kolumny *statement* (pod względem liczby słów) wynosi 17.91 ± 7.75 , co wskazuje na stosunkowo krótkie wypowiedzi. W kolumnie *justification* średnia liczba słów to 72.08 ± 64.63 , co sugeruje większe zróżnicowanie oraz dłuższą formę uzasadnień. Przypadki zawierają bardzo niewiele błędów gramatycznych — ich średnia liczba wynosi 0.38 ± 0.67 , a dla 75% przykładów (górny kwartyl) nie przekracza jednego. Średnia liczba znaków interpunkcyjnych na 100 pierwszych znaków w kolumnie *statement* wynosi 1.94 ± 1.54 , co jest zgodne biorąc pod uwagę średnią ilość słów w kolumnie *statement*.

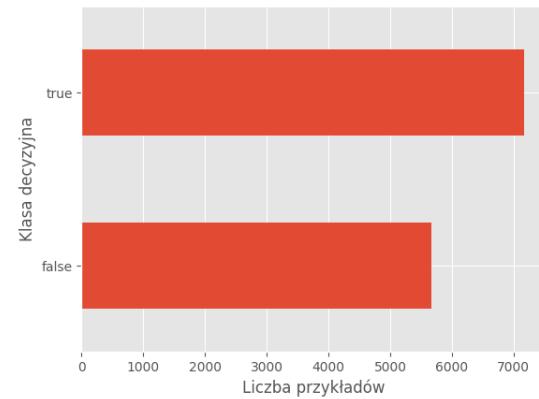
W zbiorze nie ma zduplikowanych wierszy, które by miały takie same wartości na wszystkich kolumnach. Istnieją jednak takie przykłady, które są zduplikowane na kolumnie *statement*. To są często przykłady o krótkiej, bardzo ogólnej treści. Mogą się różnić na różnych kolumnach, ale w szczególności można zauważyc, że różnią się tym, kto jest ich autorem oraz tematami, jakie poruszają. Przykładowo treść kolumny *statement* – „On abortion” – istnieje zarówno dla autorów Mitta Romney’ego oraz Charliego Crist’ego. Mitt Romney jest oznaczony jako należący do partii republikańskiej, a Charlie Crist jest oznaczony jako należący do partii demokratów. Wypowiedź Mitta Romney’ego jest przypisana do tematów ABORTION, CORRECTIONS-AND-UPDATES oraz jest oznaczona jako FALSE, a wypowiedź Charliego Crist’ego jest przypisana do tematów ABORTION, CANDIDATES-BIOGRAPHY i jest oznaczona jako HALF-TRUE. Istnieje 20 duplikatów, które mają takie same wartości na kolumnach *label* i *statement*.

Zbiór jest stosunkowo dobrze zbalansowany, ponieważ liczebność przykładów przypisanych do poszczególnych klas nie różni się zbyt mocno od siebie (Rysunki 4.1 i 4.2).

RYSUNEK 4.1: Zbalansowanie zbioru dla 6 klas decyzyjnych



RYSUNEK 4.2: Zbalansowanie zbioru dla 2 klas decyzyjnych

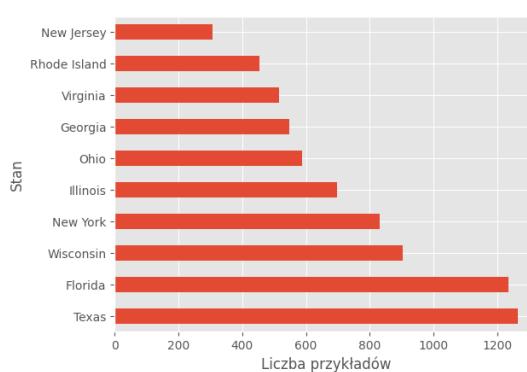


Kolumna *statement* posiada słownik o rozmiarze 13,969 tokenów, kolumna *justification* posiada słownik o rozmiarze 26,993 tokenów. Razem cały zbiór posiada słownik o rozmiarze 28,962 tokenów. W sumie 1,164,319, gdzie unikalne tokeny, które pojawiają się wiele razy, są zliczane tyle razy, ile razy się pojawiają. Po wykluczeniu tokenów *stop-words* treści posiadają słownik o rozmiarze 28,667 tokenów, a 617,537 tokenów się powtarza i 10 najpopularniejszych tokenów to: „said”, „percent”, „\$”, „says”, „state”, „nt”, „obama”, „tax”, „years” i „year”.

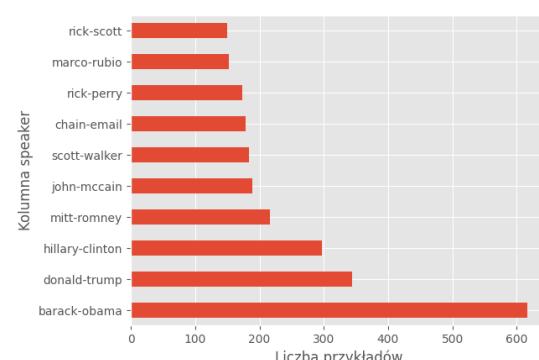
Żaden przykład nie zawiera odnośników WWW w kolumnie *statement*. W kolumnie *justification*, 6 przykładów zawiera odnośniki i wszystkie z nich są podzielone spacjami w nazwie domeny, np. „www.getourtroopsout.com”.

Jest 69 unikalnych wartości reprezentujących stany. Największa liczba wypowiedzi pochodzi z 5 najpopularniejszych stanów: Teksas, Floryda, Wisconsin, Nowy York i Illinois (Tabela 4.3). Najwięcej wypowiedzi jest od Baracka Obamy - 616, Donalda Trumpa - 344 i Hilary Clinton - 297 (Rysunek 4.4). Są 24 możliwe unikalne wartości kolumny *party-affiliation*. Najwięcej wypowiedzi jest z partii republikańskiej, potem z partii demokratów i następnie z nieoznaczonych źródeł (Rysunek 4.5). Istnieje 5,011 unikalnych wartości kolumny *context*. Składa się z różnorodnego korpusu treści. Rozmiar słownika tego korpusu posiada 2,550 unikalnych tokenów (innych niż znaki interpunkcyjne i białe znaki). Istnieje 1,360 unikalnych wartości kolumny *job_title*, a rozmiar słownika tej kolumny wynosi 1,290 unikalnych tokenów (pomijając znaki interpunkcyjne i znaki białe).

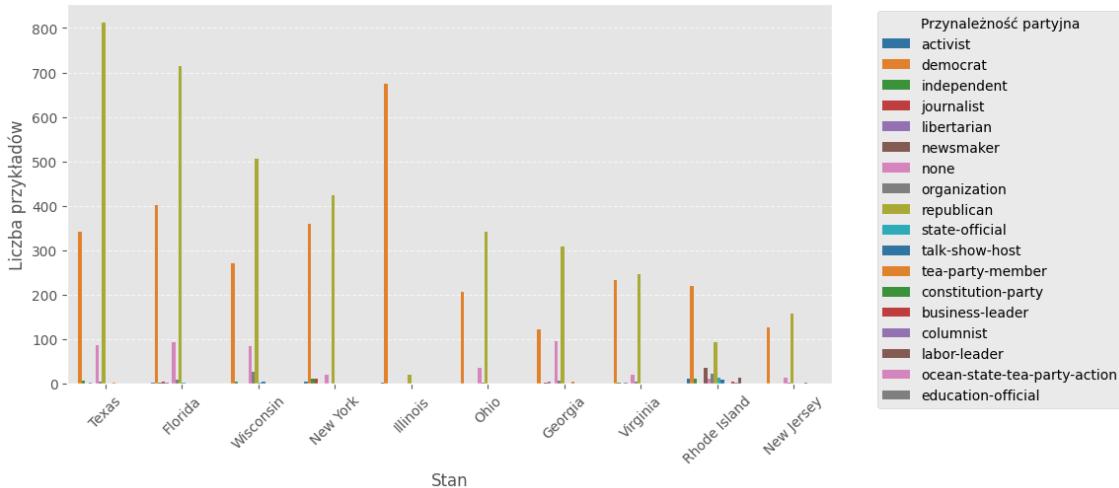
RYSUNEK 4.3: Liczba wypowiedzi w poszczególnych stanach



RYSUNEK 4.4: 10 najpopularniejszych mówców

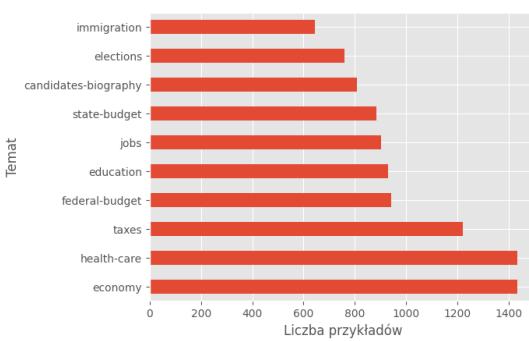


RYSUNEK 4.5: Liczba wypowiedzi na stan na przynależność partyjną (10 najpopularniejszych stanów)

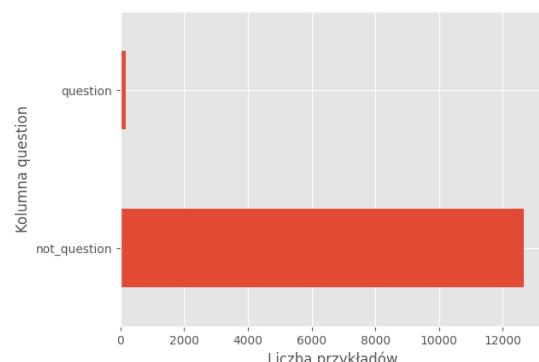


Jest 27,796 pojedynczych tematów. Pięć najpopularniejszych tematów to ekonomia, opieka zdrowotna, podatki, budżet federalny i edukacja (Rysunek 4.6). Każdy przykład może zawierać nawet kilka tematów oddzielonych od siebie przecinkiem.

RYSUNEK 4.6: 10 najpopularniejszych tematów



RYSUNEK 4.7: Typy zdań w zbiorze

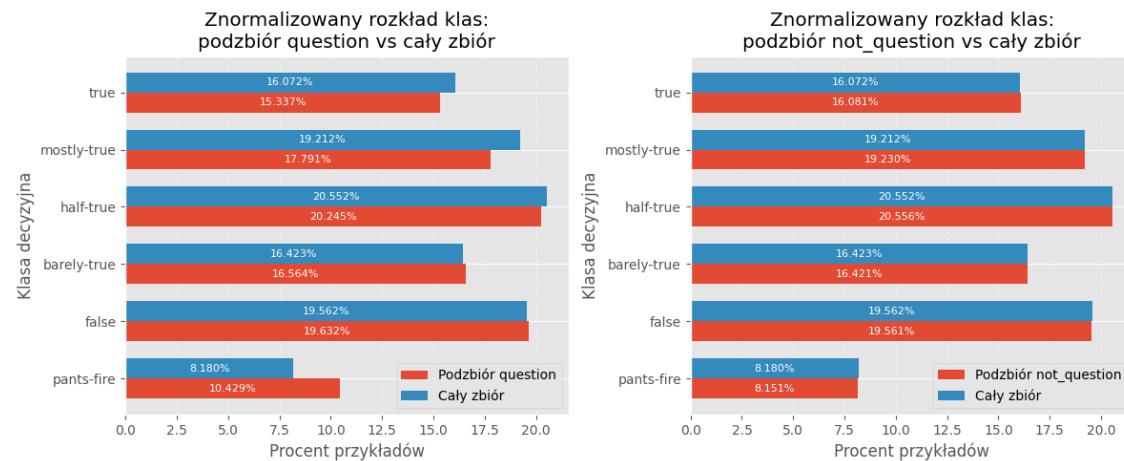


12,673 przykłady w zbiorze są zdaniemami twierdzącymi w kolumnie *statement*, a 163 jest zdaniami pytającymi (rysunek 4.7). Można zauważać, że procentowa liczba przykładów ze zdaniami pytającymi w kolumnie *statement* jest większa dla klas decyzyjnych PANTS-FIRE, FALSE i BARELY-TRUE i jednocześnie mniejsza dla klas decyzyjnych PANTS-FIRE, FALSE i BARELY-TRUE (Rysunek 4.8). W porównaniu z różnicami w zestawieniu „podzbiór question vs cały zbiór” i „podzbiór not_question vs cały zbiór”, to widać, że ten pierwszy odznacza się różnicami w stosunku do całego zbioru, podczas gdy ten drugi jest procentowo prawie identyczny w rozkładzie do całego zbioru, choć również zachowuje podobną relację, ale odwróconą.

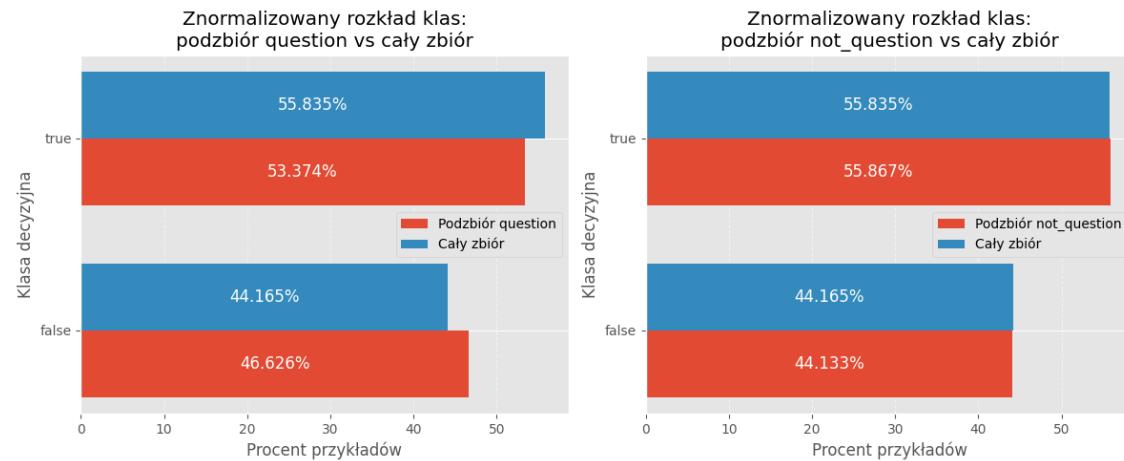
To może sugerować, że zdania pytające są częściej przypisywane do klas fałszu, ale jednocześnie trzeba pamiętać, że liczba takich przykładów jest bardzo mała i z tego powodu te różnice mogą być zauważalne, podczas gdy te w drugim zestawieniu nie są aż tak widoczne ze względu na dużo większą liczbę przykładów, chociaż też są, tylko odwrotne. Pomimo to, skoro zawsze jest więcej przykładów zaklasyfikowanych jako QUESTION w klasach falszu niż w klasach prawdy, to można założyć, że ten trend nie jest przypadkowy.

Mogą to zaobserwować też na uogólnionych dwóch klasach decyzyjnych (Rysunek 4.9). Na tym rysunku procentowa liczba przykładów jest większa w podzbiorze NOT_QUESTION w klasie TRUE

RYSUNEK 4.8: Procentowy rozkład klas w stosunku do podzbiorów question i not_question oraz procentowej liczby przykładów w całym zbiorze (6 klas decyzyjnych)



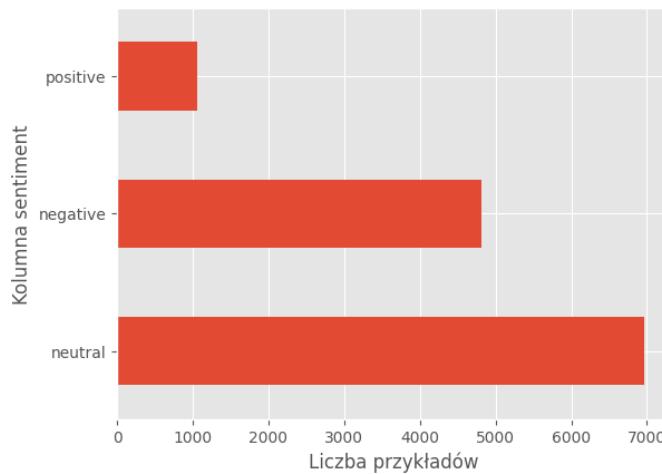
RYSUNEK 4.9: Procentowy rozkład klas w stosunku do podzbiorów question i not_question oraz procentowej liczby przykładów w całym zbiorze (2 klasy decyzyjne)



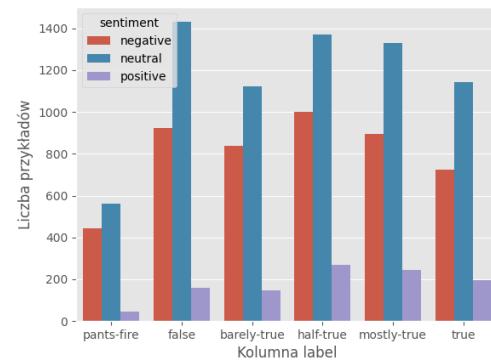
w stosunku do całego zbioru niż w klasie FALSE. Widać tą zależność też na rysunku z sześcioma klasami decyzyjnymi.

Porównując kolumnę *label* i *sentiment* można zauważyc, że w każdej klasie w wypowiedziach przeważa neutralny sentyment, później negatywny i na końcu pozytywny (Rysunek 4.10). Sentyment pozytywny jest średnio większy w ogólnej klasie TRUE (Rysunek 4.11). Najwięcej jest wypowiedzi o neutralnym sentymencie, później negatywnym i na końcu pozytywnym (Rysunek 4.12).

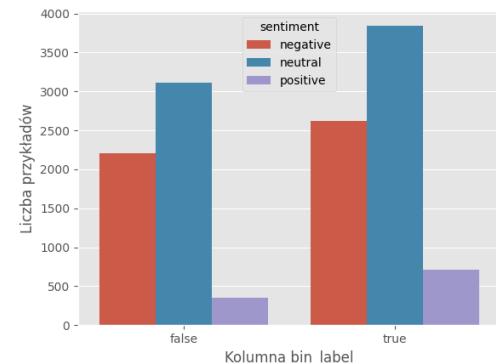
RYSUNEK 4.12: Rozkład sentymentu w zbiorze



RYSUNEK 4.10: Zestawienie kolumn label vs sentiment (6 klas decyzyjnych)

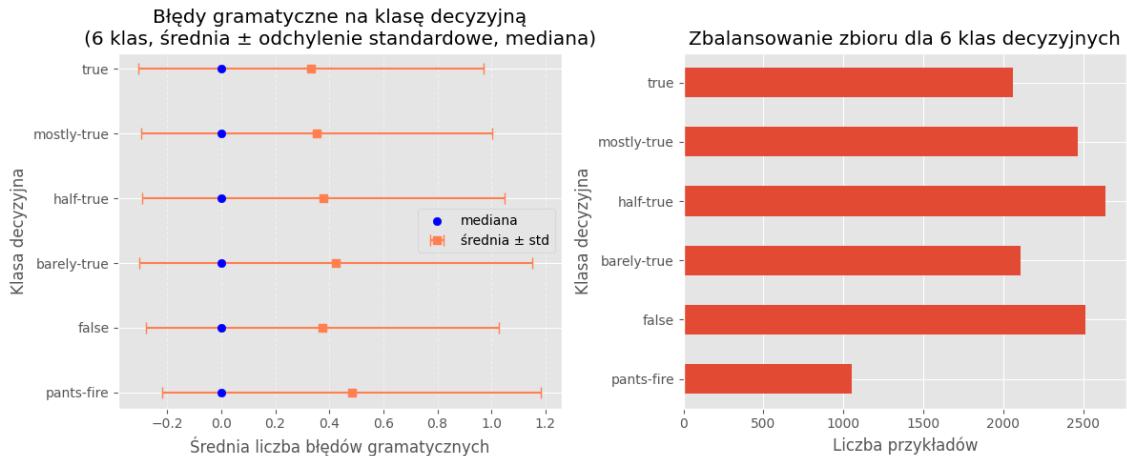


RYSUNEK 4.11: Zestawienie kolumn binlabel vs sentiment (2 klasy decyzyjne)

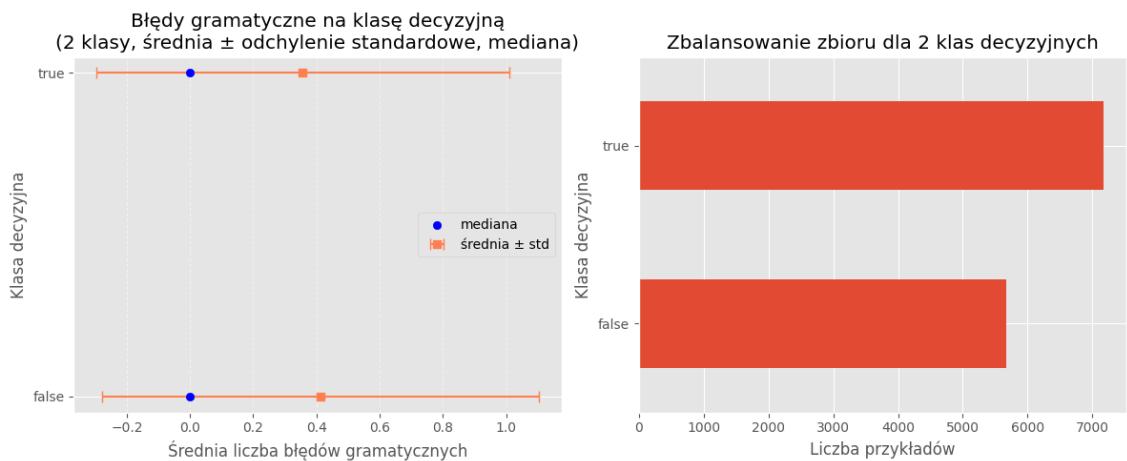


Średnia liczba błędów gramatycznych w poszczególnych klasach decyzyjnych jest większa dla PANTS-FIRE, BARELY-TRUE, i niewiele też brakuje dla klasy FALSE, w analizie sześcioklasowej niż dla klas HALF-TRUE, MOSTLY-TRUE i TRUE, pomimo że liczba przykładów w klasach HALF-TRUE, MOSTLY-TRUE i TRUE jest większa niż w klasach PANTS-FIRE, FALSE i BARELY-TRUE (Rysunek 4.13). Taka sama relacja jest zachowana w analizie dwuklasowej (Rysunek 4.14). To może sugerować, że przykłady z błędami gramatycznymi są częściej klasyfikowane jako fałszywe. Przeważającymi nad resztą rodzajami błędów gramatycznych są możliwy błąd w pisowni i niepoprawna pisownia form ściągniętych w języku angielskim. Te rodzaje błędów ma po ponad 1,000 przykładów, które je posiadają, podczas gdy inne rodzaje błędów pojawiają się w mniej niż 100 przykładach (Tabela 4.2).

RYSUNEK 4.13: Błędy gramatyczne na klasę decyzyjną + zbalansowanie zbioru dla porównania (6 klas decyzyjnych)



RYSUNEK 4.14: Błędy gramatyczne na klasę decyzyjną (2 klasy decyzyjne)

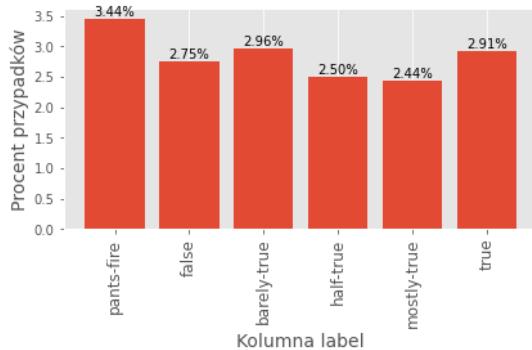
TABELA 4.2: 10 najpopularniejszych błędów gramatycznych wykrytych w kolumnie *statement* w zbiorze train2.tsv

Nazwa zasady	Opis znaczenia	Liczba wystąpień
MORFOLOGIK_RULE_EN_US	Możliwy błąd w pisowni	1901
EN_CONTRACTION_SPELLING	Niepoprawna pisownia form ściągniętych w języku angielskim	1028
COMMA_PARENTHESIS_WHITESPACE	Użycie białych znaków przed przecinkiem lub przed albo za nawiasem	83
IT_IS	Its zamiast it's	70
POSSESSIVE_APOSTROPHE	Może brakować apostrofu	66
COMMA_COMPOUND_SENTENCE	Przecinek między niezależnymi klauzulami	58
UPPERCASE_SENTENCE_START	Sprawdza, czy zdanie zaczyna się od wielkiej litery	56
CANT	Cant zamiast can't	51
IVE_CONTRACTION	Ive zamiast I've	44
EN_UNPAIRED_QUOTES	Brak otwarcia lub zamknięcia dla znaków cytowania	43

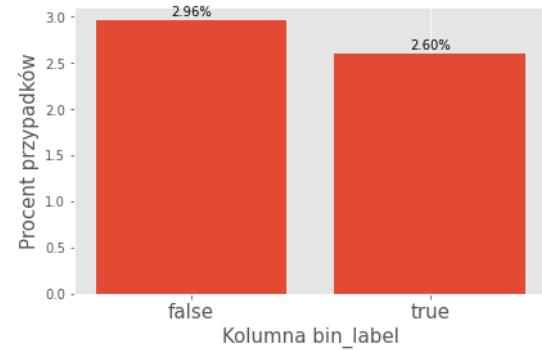
W całej kolumnie *curse* tylko 2.8% przypadków zostało zaklasyfikowanych jako wulgarne. Oznacza to, że bardzo rzadko klasyfikator, który został użyty do wygenerowania tej kolumny klasyfikuje *statement* jako wulgarny. Analizując procent występowania wulgarnych wypowiedzi w klasyfikacji sześcioklasowej, można zauważać, że generalnie im wypowiedź jest bardziej fałszywa, tym wię-

sza szansa, że jest zaklasyfikowana jako wulgarna (Rysunek 4.15). W analizie dwuklasowej więcej wypowiedzi z klasy decyzyjnej FALSE jest oznaczona jako wulgarna (Rysunek 4.16). Może to wskazywać na to, że jeżeli *statement* jest klasyfikowany jako wulgarny, to jest zwiększoną szansą na to, że będzie fałszywy.

RYSUNEK 4.15: Procent występowania wulgarnych wypowiedzi vs label (6 klas decyzyjnych)

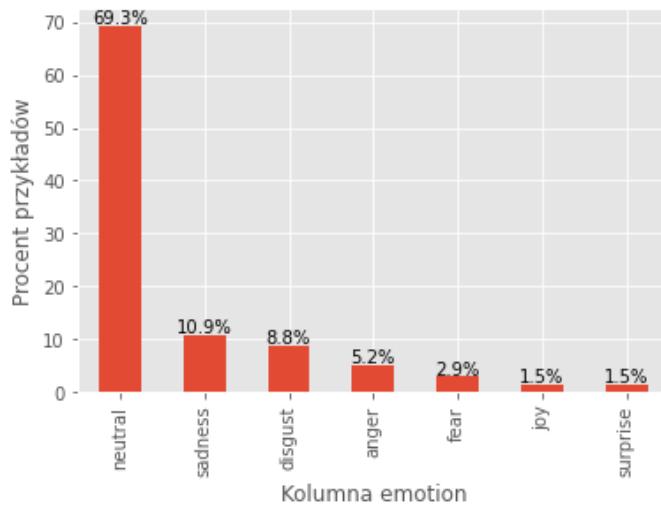


RYSUNEK 4.16: Procent występowania wulgarnych wypowiedzi vs bin_label (2 klasy decyzyjne)

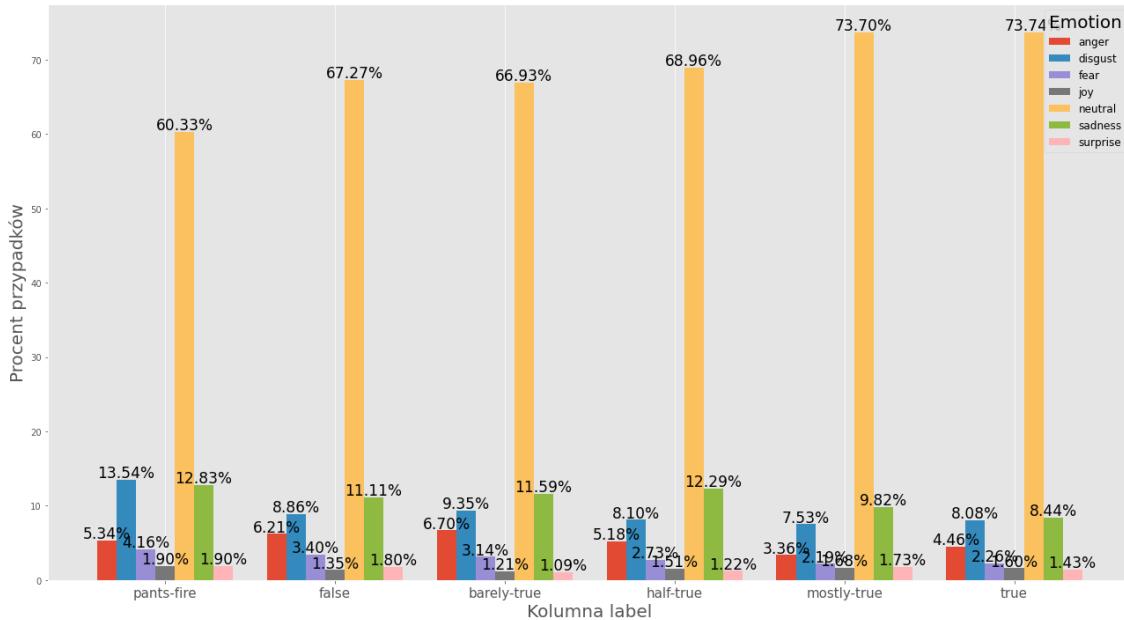


Większość wypowiedzi w kolumnie *emotion* została zaklasyfikowana do klasy "neutral". Oznacza to, że większość wypowiedzi ma nacechowanie emocjonalne neutralne. (Rysunek 4.17). Dla klasyfikacji sześcioklasowej następuje wzrost nacechowania neutralnego wraz z poziomem prawdziwości. Emocja "disgust" najczęściej występuje dla klasy decyzyjnej PANTS-FIRE (13.54%, gdzie dla drugiej pod względem liczności jest to 9.35% przypadków). Dla pozostałych klas decyzyjnych różnica nie jest tak duża. Etykieta "sadness" występuje na podobnym poziomie w klasach decyzyjnych PANTS-FIRE, FALSE, BARELY-TRUE, HALF-TRUE. Dla dwóch najbardziej prawdziwych klas MOSTLY-TRUE i TRUE występowanie tego nacechowania emocjonalnego jest mniejsze. Etykiety "fear" i "anger" stopniowo zwiększały swój udział dla wypowiedzi, które są bardziej fałszywe. Nacechowanie emocjonalne "joy" i "surprise" występuje na podobnym poziomie dla wszystkich klas decyzyjnych. (Rysunek 4.18). W klasyfikacji dwuklasowej rozkład wartości *emotion* zachowuje się podobnie jak w klasyfikacji sześcioklasowej (Rysunek 4.19).

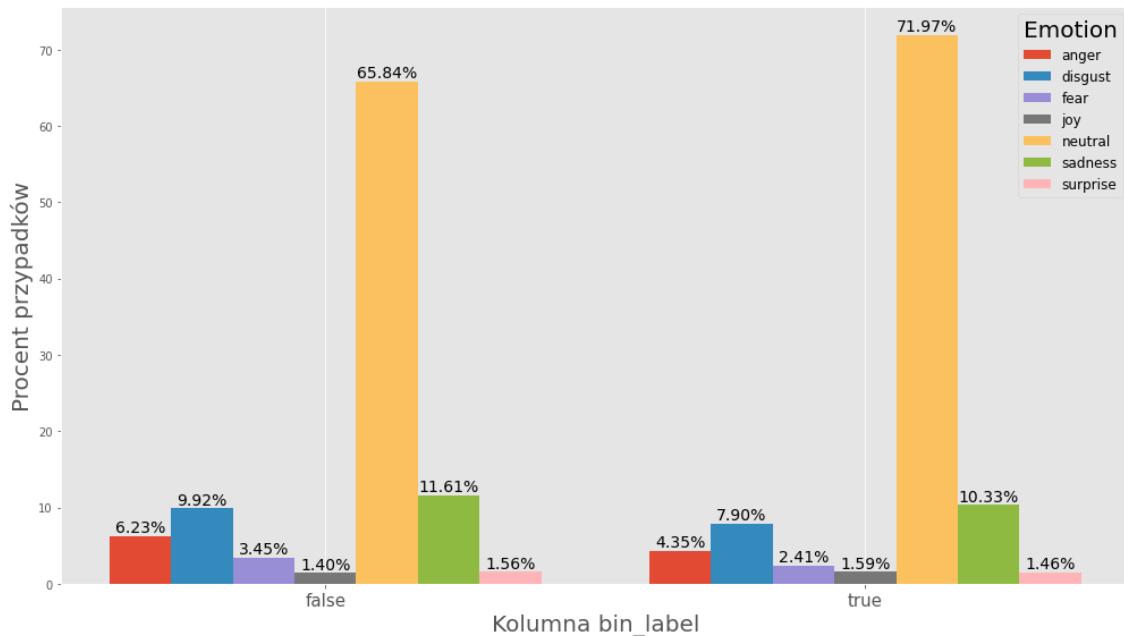
RYSUNEK 4.17: Rozkład nacechowania emocjonalnego w zbiorze



RYSUNEK 4.18: Procent występowania etykiet emotion vs label (6 klas decyzyjnych)

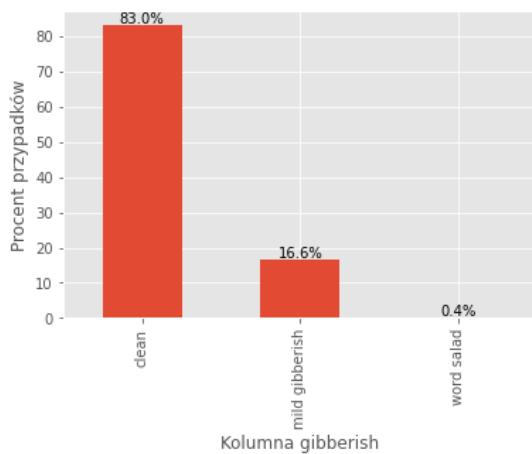


RYSUNEK 4.19: Procent występowania etykiet emotion vs bin_label (2 klasy decyzyjne)

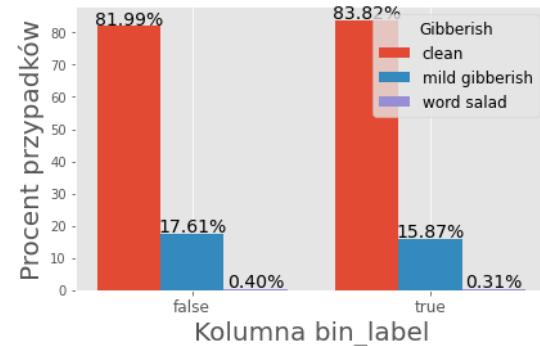


Większość wypowiedzi w kolumnie *gibberish* została sklasyfikowana jako "clean" (Rysunek 4.20). W klasyfikacji sześcioklasowej wraz ze wzrostem prawdziwości wypowiedzi, wzrasta udział etykiety "clean", a spada udział etykiety "mild gibberish". Etykieta "word salad" występuje na podobnych poziomach we wszystkich klasach decyzyjnych, jednak w bardzo małych ilościach. W klasie decyzyjnej FALSE jest jej najwięcej - 0.8%, a w klasie PANTS-FIRE nie wystąpiła ani jedna. Warto zauważyć, że użyty do generowania tej kolumny klasyfikator ma jeszcze dostępną etykię "noise", jednak nie została ona przyporządkowana do ani jednej wypowiedzi, więc nie ma jej w analizie. Rozkład wartości w tej kolumnie może wskazywać na to, że fałszywe wypowiedzi są pisane mniej skomplikowanym językiem (Rysunek 4.22). W klasyfikacji dwuklasowej także dla klasy decyzyjnej TRUE jest większy udział etykiety "clean", a mniejszy udział etykiety "mild gibberish" (Rysunek 4.21).

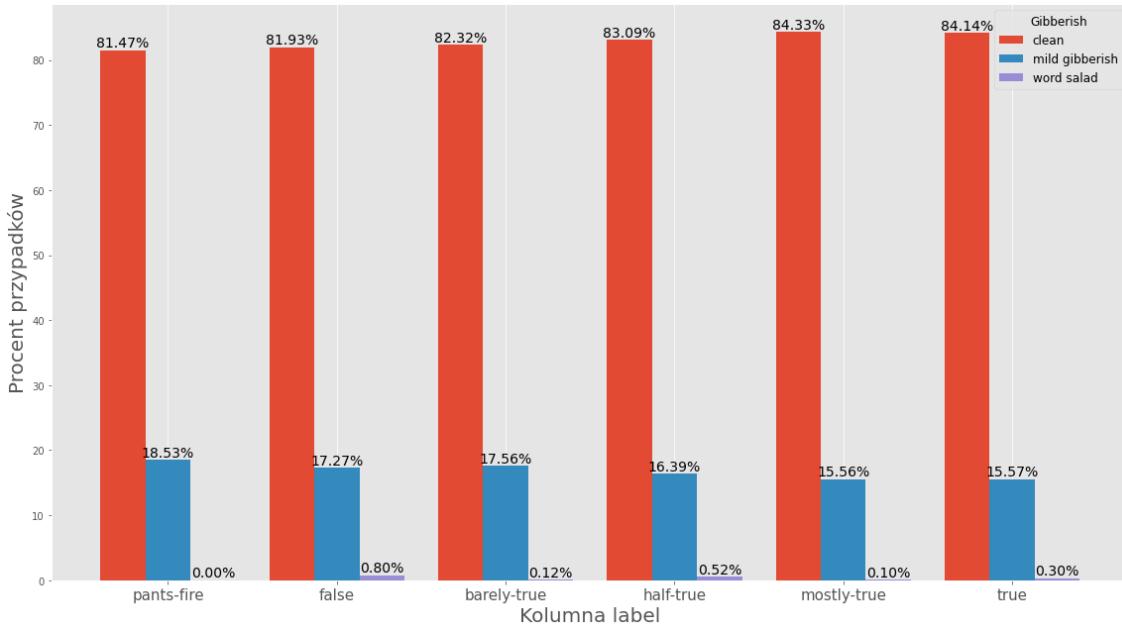
RYSUNEK 4.20: Rozkład wartości w kolumnie gibberish



RYSUNEK 4.21: Procent występowania etykiet gibberish vs bin_label (2 klasy decyzyjne)

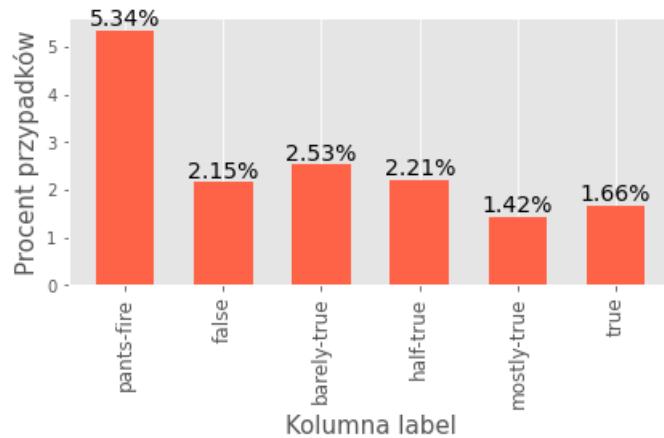


RYSUNEK 4.22: Procent występowania etykiet gibberish vs label (6 klas decyzyjnych)



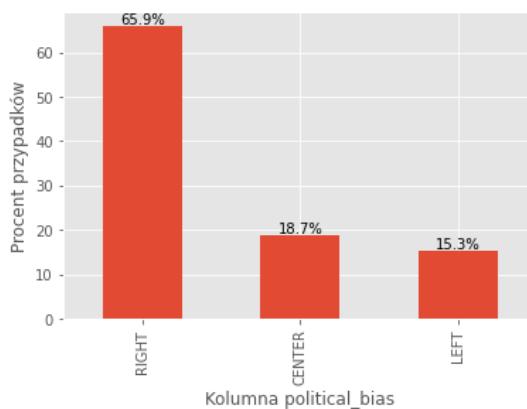
W kolumnie *offensiveness* tylko 2.3% zostało zaklasyfikowane do etykiety "offensive". W klasyfikacji sześcioklasowej etykieta ta, występuje najczęściej dla klasy decyzyjnej PANTS-FIRE - 5.34%. Dla pozostałych klas poziom występowania waha się od 1.42%, do 2.53%. Dla dwóch najbardziej prawdziwych klas decyzyjnych MOSTLY-TRUE i TRUE etykieta "offensive" występuje najrzadziej (Rysunek 4.23). W klasyfikacji dwuklasowej poziom występowania wypowiedzi oznaczonych jako "offensive" pomiędzy klasami decyzyjnymi jest podobny. 2.89% dla FALSE i 1.78% dla TRUE.

RYSUNEK 4.23: Procent występowania wypowiedzi oznaczonych etykieta "offensive" w kolumnie offensiveness vs label (6 klas decyzyjnych)

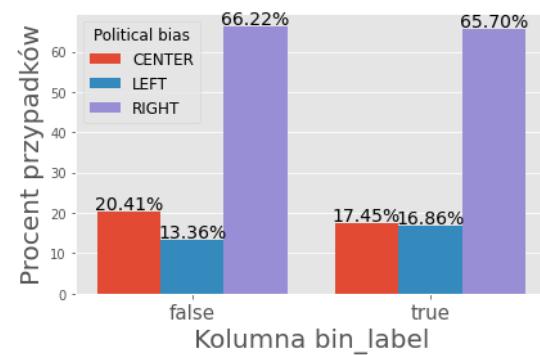


Większość wartości w kolumnie *political_bias* stanowi etykieta "right" 65.9% (Rysunek 4.24). Wynika z tego, że większość wypowiedzi została zaklasyfikowana jako prawicowe. Liczba wypowiedzi centrowych i lewicowych jest podobna, z delikatną przewagą centrowych. W klasyfikacji sześcioklasowej procent wypowiedzi prawicowych jest największy dla klas PANTS-FIRE i TRUE, jednak jest dosyć wyrównany dla wszystkich klas decyzyjnych (najwięcej 67.97% dla TRUE i najmniej 64.21% dla BARELY-TRUE). Trzy największe wartości procentowe wypowiedzi lewicowych są dla trzech najbardziej prawdziwych klas decyzyjnych, dodatkowo najmniejszy procent wypowiedzi lewicowych jest dla klasy PANTS-FIRE (Rysunek 4.26). Procent wypowiedzi centrowych największy jest w klasie decyzyjnej PANTS-FIRE, zauważalny jest trend zmniejszającego się udziału wypowiedzi centrowych, wraz z postępem prawdziwości klasy decyzyjnej. W klasyfikacji dwuklasowej procent wypowiedzi prawicowych jest podobny dla obydwu klas decyzyjnych. Dodatkowo w klasie decyzyjnej TRUE większy jest procent wypowiedzi lewicowych i mniejszy jest udział wypowiedzi centrowych w odniesieniu do klasy decyzyjnej FALSE.

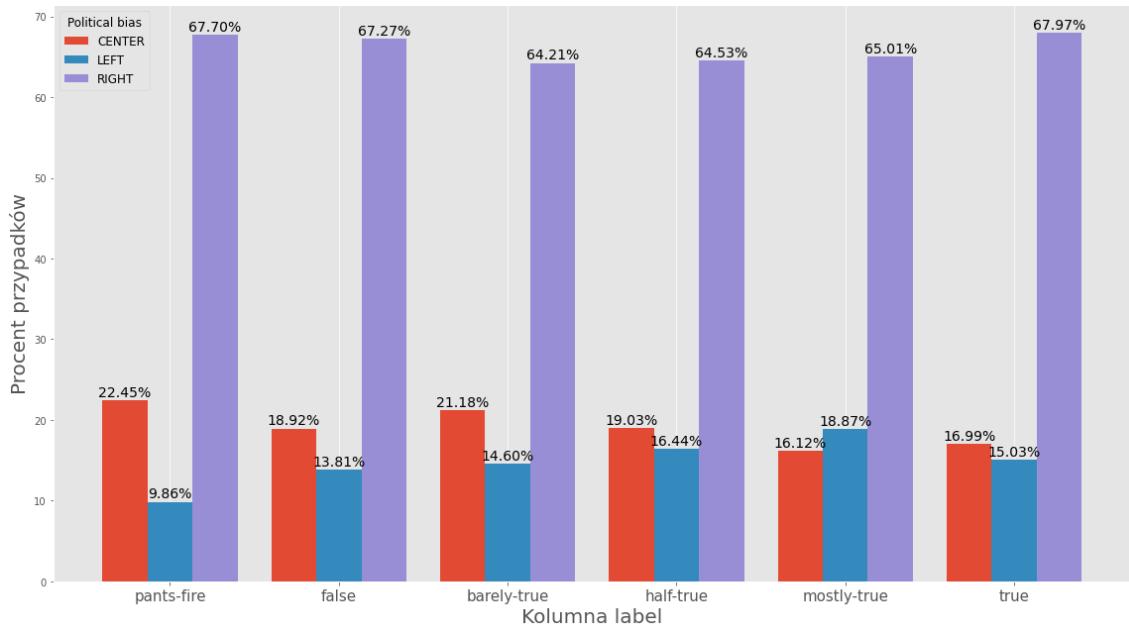
RYSUNEK 4.24: Rozkład wartości w kolumnie political_bias



RYSUNEK 4.25: Procent występowania etykiet political_bias vs bin_label (2 klasy decyzyjne)



RYSUNEK 4.26: Procent występowania etykiet political_bias vs label (6 klas decyzyjnych)



Badaliśmy też inne zależności, ale nie znaleźliśmy niczego interesującego.

4.4 Normalizacja zbioru danych

Przed trenowaniem modeli dokonano normalizacji zbiorów. Poddane temu procesowi zostały pliki `train2.tsv`, `val2.tsv` i `test2.tsv`. Metodą normalizacji jest wykorzystanie klasy z pakietu `sklearn.preprocessing` – `StandardScaler`, który działa na zasadzie obliczenia średniej wartości kolumny i odchylenia standardowego na wybranym zbiorze, a następnie stosuje wzór:

$$z = \frac{x - \mu}{\sigma}$$

, gdzie:

- x - Wartość oryginalnej cechy.
- μ - Średnia wartość cechy.
- σ - Odchylenie standardowe cechy.
- z - Znormalizowana wartość cechy.

Parametry normalizacji (μ i σ) zostały wyznaczone na zbiorze `train2.tsv`, a następnie przy ich pomocy zostały znormalizowane wybrane kolumny w zbiorach wspomnianych powyżej. Wybranymi kolumnami ze zbiorów są: `barely_true_counts`, `false_counts`, `half_true_counts`, `mostly_true_counts`, `pants_on_fire_counts`, `grammar_errors`, `ratio_of_capital_letters`.

Następnie znormalizowane zbiory zostały ponownie zapisane, ale już do innych plików w formacie CSV (zamiast TSV): `train2.csv`, `val2.csv` i `test2.csv`.

Przy okazji normalizacji poprawione zostały również wartości kolumny `curse`. Wszystkie wartości tej kolumny, które są równe "Curse" zamieniane są na zapisane z małej litery "curse", a wszystkie wartości `NaN` zamieniane są na "non-curse".

We wszystkich dalszych procesach związanych z trenowaniem była wykorzystywana ta wersja zbiorów.

Rozdział 5

Rozwiązanie

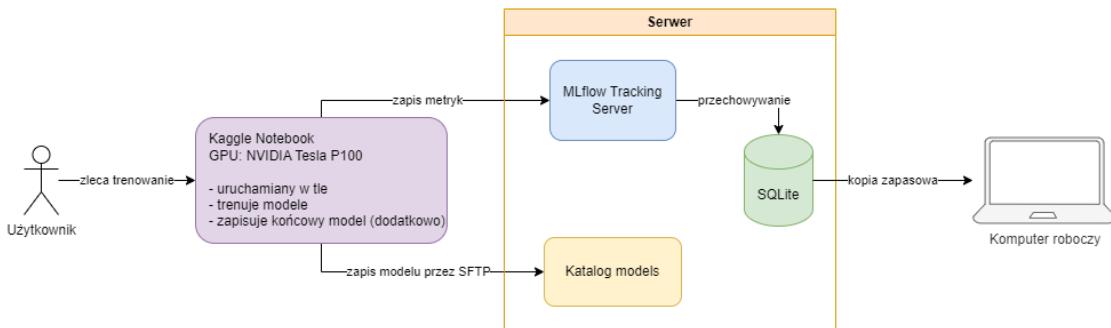
W celu realizacji zadania postawionego w rozdziale 1.2, zdecydowaliśmy się zaimplementować model klasyfikatora przy pomocy modelu RoBERTa, który został opisany w rozdziale 2.8.2. Opisujemy podejście do trenowania modeli w całym projekcie oraz architekturę naszego procesu treningu. Z powodu użycia kilku kolumn tekstowych, które nie mogłyby zostać zakodowane w postaci kodowania one-hot, tylko wymagały konwersji na postać wektorów zanurzeń (ang. word embeddings), mieliśmy trzy możliwe ścieżki do obranego. Zbadaliśmy każdą z nich pod kątem dokładności oraz miary F1 na zbiorze walidacyjnym, a także pod kątem optymalizacji wymaganych obliczeń. W wyniku tych eksperymentów wybraliśmy jedną ze ścieżek i na niej budowaliśmy dalej nasze rozwiązanie. Przed zbadaniem wariantów implementacji klasyfikatora sprawdziliśmy również, czy udało nam się zbliżyć do wyników autorów zbioru LIAR PLUS poprzez najprostszy możliwy wariant klasyfikatora z użyciem modelu RoBERTa.

5.1 Proces treningu

Do trenowania modeli wykorzystaliśmy platformę Kaggle. Umożliwia ona uruchamianie interaktywnych notatników (**Jupyter Notebook**), w których można zapisywać instrukcje, a następnie je uruchamiać. Kaggle pozwala użytkownikom na wykorzystywanie jednej z kilku dostępnych opcji akceleracji obliczeń przy pomocy karty graficznej lub innego rodzaju akceleratora. Dostępne akceleratory to:

- **NVIDIA T4 x2** - karta graficzna z serii Tensor Core GPU, reklamowana jako produkt przeznaczony do inferencji (czyli wykonywania już wytrenowanych modeli). Oznaczenie „x2” oznacza, że środowisko Kaggle udostępnia w sesji dwie takie karty równocześnie, co zwiększa moc obliczeniową. Została zbudowana na architekturze Turing. Ma 16 GB pamięci VRAM GDDR6 i wspiera obliczenia FP32, FP16, INT8, INT4 oraz Tensor Cores (do akceleracji obliczeń macierzowych), których jest 320 [12, 2].
- **NVIDIA Tesla P100** - wysokowydajna karta graficzna z serii Pascal, zaprojektowana specjalnie do zastosowań obliczeniowych w centrach danych, zarówno do obliczeń wysokiej wydajności (*High Performance Computing* - HPC) jak i do obliczeń AI. Jest mocniejsza niż T4 w niektórych zastosowaniach treningowych. Jest sprzedawana w dwóch wersjach - 16 GB lub 12 GB pamięci HBM2. Jest opisywana jako przeznaczona głównie do trenowania modeli głębokiego uczenia [13, 2].
- **TPU VM v3-8** - specjalny typ procesora zaprojektowany przez Google wyłącznie do przyspieszania obliczeń w TensorFlow. Oznaczenie „v3-8” oznacza jednostkę trzeciej generacji

RYSUNEK 5.1: Architektura procesu treningu



złożoną z 8 rdzeni TPU (czyli tzw. chipów), które działają równolegle. Ta technologia jest opisywana w artykule Rukshara jako najbardziej wydajna [2]. W praktyce v3-8 może przewyższać P100 czy T4, ale jej pełna moc jest najlepiej wykorzystywana w dużych modelach i optymalnie przygotowanym kodzie [29].

We wszystkich etapach trenowania modeli wykorzystaliśmy kartę graficzną NVIDIA Tesla P100, której właściwości obliczeniowe okazały się najlepiej dopasowane do naszych potrzeb. Obliczenia były wykonywane na rdzeniach CUDA.

Do zbierania metryk podczas treningu wykorzystaliśmy narzędzie **MLflow** i skonfigurowaliśmy je na osobnym komputerze, który będziemy dalej nazywać **serwerem**, ponieważ służył również jako serwer, na którym zapisywaliśmy pośrednio zrzuty modelu podczas procesu trenowania oraz finalne wersje modeli. Wszystkie ustawienia oraz eksperymenty są zapisywane w bazie danych **SQLite**, którą co jakiś nieokreślony czas kopiowaliśmy na nasze komputery robocze, aby zabezpieczyć się przed potencjalną awarią.

Zapisywanie modeli odbywało się dzięki bibliotece **paramiko**, która umożliwia między innymi transfer plików poprzez SFTP. Zapisywaliśmy modele na naszym serwerze w wybranym katalogu.

Aby to wszystko mogło funkcjonować, to wykorzystaliśmy funkcję „Save Version” w Kaggle, która umożliwia uruchamianie notatników w tle, dzięki czemu można było po uruchomieniu złożyć zadanie na wiele godzin i po zakończeniu sprawdzić wyniki lub pobrać gotowe modele. Kaggle również pozwala przechowywać pliki z sesji, dzięki czemu mieliśmy gotowe modele nie tylko w naszym serwerze, ale również na platformie Kaggle, co jest dodatkowym zabezpieczeniem.

Nie wykorzystaliśmy funkcji zapisu modeli przy pomocy biblioteki MLflow, ponieważ ta opcja działa tylko, gdy uruchomi się usługę MLflow lokalnie. Gdy próbuje się zapisać model zdalnie na własnym serwerze, to biblioteka MLflow i tak tworzy kopię pliku lokalnie w wyznaczonym przez siebie katalogu.

Architekturę procesu treningu można zobaczyć na Rysunku 5.1.

5.2 Sposób trenowania

Funkcja trenująca posiada główną pętle, która iteruje po epokach, a wewnętrznie niej są dwie pętle, gdzie pierwsza iteruje po batchach ze zbioru treningowego a druga po batchach ze zbioru walidacyjnego. Pierwsza z nich trenuje model i zbiera pomiary na zbiorze treningowym, a druga zapisuje pomiary modelu na zbiorze walidacyjnym.

5.2.1 Zbieranie metryk

Na zbiorze treningowym mierzone są: średnia dokładność modelu, średnia wartość loss (czyli średnia wartość funkcji straty), wartość miary F1, wartość precyzyji, wartość pełności. F1, precyza i pełność są liczone z uśrednieniem macro. Analogicznie postępujemy ze zbiorem walidacyjnym. Zapisujemy również miarę F1, precyzyję i pełność dla każdej osobnej klasy. Do zbierania pomiarów F1, precyzyji i pełności wykorzystaliśmy bibliotekę `torchmetrics`.

5.2.2 Mechanizm wcześniego zatrzymywania

Jeśli średnia dokładność modelu na zbiorze walidacyjnym nie wzrosła przez ustaloną ilość epok, którą sterujemy parametrem `patience`, to proces trenowania jest zatrzymywany.

5.2.3 Mechanizm zapisu kroków pośrednich

Co ustaloną liczbę epok, sterowaną parametrem `checkpoint_send_interval` wysyłamy zapis aktualnej wersji modelu do wybranego katalogu modelu na naszym serwerze. Nazywamy ten plik `checkpoint_(patience).pth`, gdzie „(patience)” jest wartością parametru `patience`. W ten sposób oddzielamy pośrednio zapisy, które nazywamy krokami pośrednimi, od najlepszych wersji modelu, które zapisujemy z nazwą `best_model_(patience).pth` oraz oszczędzamy czas na ciągłym przesyłaniu pliku po sieci. Plik `checkpoint_(patience).pth` służy jako miejsce, od którego można zacząć dalszy trening, np. w przypadku awarii programu.

W pliku `checkpoint_(patience).pth` zapisujemy:

- Parametry obiektu modelu.
- Stan obiektu optymalizatora.
- Numer aktualnej epoki.
- Dokładność na zbiorze walidacyjnym tego modelu.

Analogicznie w pliku `best_model_(patience).pth` zapisujemy te same dane. Jednocześnie warto zaznaczyć, że nie zapisujemy wszystkich parametrów (wag) modelu, a tylko te, które są zdefiniowane przez funkcję `state_for_save(self)`, która jest zdefiniowana w klasie modelu. W przypadku modelu RoBERTa mamy również dodatkowy warunek, który wybiera do zapisu tylko te parametry, które nie są zamrożone. Do odczytu parametrów z obiektu, wczytanego z pliku `*.pth`, służy funkcja `load_state_from_save(self, state)`, która również jest zdefiniowana w klasie modelu.

Na końcu procesu trenowania jest zapisywany plik `checkpoint_(patience).pth` - lokalnie i na naszym serwerze.

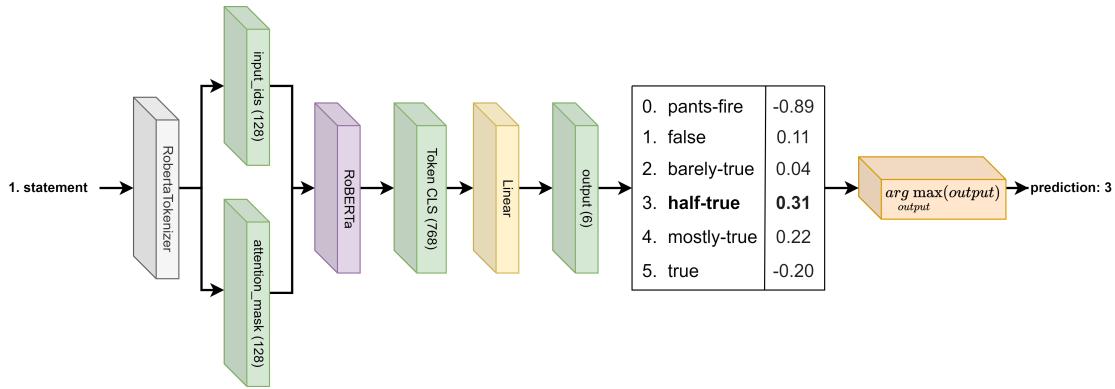
5.2.4 Optymalizator

Każdy eksperyment wykorzystuje optymalizator parametrów Adam, opisany w rozdziale 2.5.1, z ustaloną w eksperymencie wartością `lr` oznaczającą learning rate - η . W niektórych eksperymen-tach definiujemy dodatkowy learning rate dla parametrów w modelu RoBERTa.

5.2.5 Funkcja straty

Do obliczania funkcji straty wykorzystujemy w każdym eksperymencie funkcję entropii krzyżowej zdefiniowaną w rozdziale 2.9.6.

RYSUNEK 5.2: Architektura bazowego modelu bez uwzględnienia mini-batchowania przez DataLoader



5.3 Bazowa implementacja klasyfikatora RoBERTa

5.3.1 Architektura

Architektura podstawowego klasyfikatora zakłada, że jedynym wejściem modelu jest treść zawarta w kolumnie *statement*, która zostaje zamieniona przez tokenizator na tensor identyfikatorów tokenów (`input_ids`) oraz tensor maski uwagi (`attention_mask`).

Wykorzystaliśmy implementację modelu RoBERTa oraz dedykowanego tokenizatora, która jest dostępna na platformie Hugging Face [22]. Użycie modelu i tokenizatora jest możliwe dzięki paczce `transformers` [38].

Tokenizator wykorzystuje parametry `truncation = True`, `padding = max_length`, gdzie parametr `max_length = 128`, dzięki czemu wejściowy tekst *statement* jest przycinany do 128 tokenów, a jeśli treść jest krótsza, to brakujące tokeny są wypełniane specjalnymi tokenami paddingu.

Następnie RoBERTa, której wagie wewnętrzne są zamrożone zarówno w procesie treningu jak i ewaluacji, przyjmuje `input_ids` oraz `attention_mask` jako wejście i jako wyjście wykorzystywany jest wektor ukryty odpowiadający tokenowi [CLS], czyli pierwszy wektor z ostatniej ukrytej warstwy, o rozmiarze 768 składowych (implementacyjnie reprezentacją wyjściową jest tensor o wymiarach (`batch_size = 64, hidden_size = 768`)).

Tensor tokenu CLS następnie trafia do zdefiniowanej przez nas warstwy liniowej `fc`, bez funkcji aktywacji, która na wyjściu zwraca 6 logitów, reprezentujących odpowiednie prawdopodobieństwa przynależności przykładu do jednej z sześciu zdefiniowanych klas: PANTS-FIRE, FALSE, BARELY-TRUE, HALF-TRUE, MOSTLY-TRUE, TRUE. Logity te mogą przyjąć dowolną wartość ze zbioru \mathbb{R} , a dopiero po zastosowaniu funkcji softmax można otrzymać właściwy rozkład prawdopodobieństw.

Wynikiem działania modelu jest identyfikator wartości maksymalnej w wyjściowym tensorze y warstwy `fc`, a $prediction = \arg \max(y)$.

Za konwersję nazwy klasy z kolumny *label* na wartość liczbową odpowiada zmienna słownika `LABEL_MAPPING`, która przyjmuje takie wartości:

- PANTS-FIRE: 0
- FALSE: 1
- BARELY-TRUE: 2
- HALF-TRUE: 3
- MOSTLY-TRUE: 4

- TRUE: 5

Graficzny opis architektury jest przedstawiony na Rysunku 5.2. Oczywiście dokładne działanie modelu jest bardziej złożone ze względu na chociażby wykorzystanie mini-batchowania, dzięki czemu do faktycznego kształtu tensorów należałoby uwzględnić rozmiar batcha, czego nie zrobiliśmy dla ułatwienia zrozumienia głównej idei działania architektury.

5.3.2 Trenowanie

TABELA 5.1: Miary na zbiorze walidacyjnym wytrenowanego modelu bazowego

Liczba przykładów	1284
Dokładność (accuracy)	0.27
Wartość funkcji straty (loss)	1.71
F1	0.19
Precyza (precision)	0.26
Pelność (recall)	0.23

TABELA 5.2: Porównanie dokładności modeli na zbiorze walidacyjnym, opisanych w artykule LIAR PLUS dla wariantu S, w sześcioklasowej klasyfikacji, z modelem bazowym

Model	Dokładność
LR	0.23
SVM	0.25
BiLSTM	0.26
Model bazowy	0.27

Proces trenowania był ograniczony parametrem `patience = 10` i maksymalną liczbą epok równą 30. Ostatecznie trenowanie nie zostało przerwane wcześniej i proces trenowania trwał 30 epok. Wytrenowany model osiągnął dokładność na poziomie wartości 0.27 i co ciekawe jest to wyższy wynik niż każdy z modeli typu S, w sześcioklasowej klasyfikacji, opisanych w zbiorze LIAR PLUS (Tabele 5.1 i 5.2) [3].

TABELA 5.3: Szczegółowe miary precyzji, pełności i F1, uśrednione metodą macro, na zbiorze walidacyjnym wytrenowanego modelu bazowego oraz modeli LR i BiLSTM z artykułu o LIAR PLUS (zamieniono ważone F1 dla modeli twórców LIAR PLUS na macro)

Klasa	Rozmiar klasy	Precyza	Pelność	F1	F1 LR	F1 BiLSTM
pants-fire	116	0.00	0.00	0.00	0.18	0.19
false	263	0.28	0.50	0.36	0.28	0.34
barely-true	237	0.33	0.07	0.11	0.21	0.13
half-true	248	0.24	0.38	0.29	0.22	0.28
mostly-true	251	0.29	0.42	0.34	0.23	0.33
true	169	0.40	0.04	0.07	0.22	0.18
suma/średnia	1284	0.26	0.23	0.19	0.22	0.24

Badając zebrane miary modelu bazowego w Tabeli 5.3, można zauważyć, że przykłady w klasie PANTS-FIRE są ignorowane, ponieważ zarówno precyza i pełność są równe zero. Oznacza to, że model nie był w stanie poprawnie zaklasyfikować ani jednego przykładu jako PANTS-FIRE.

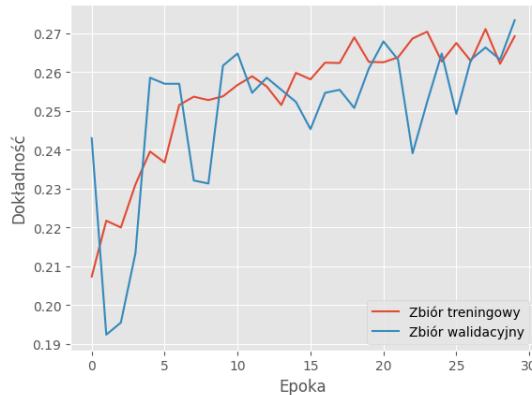
Klasy TRUE i BARELY-TRUE osiągnęły najniższe wartości miary F1 (odpowiednio 0.07 i 0.11). Niska pełność (odpowiednio 0.04 i 0.07) oznacza zaklasyfikowanie małego odsetka wszystkich poprawnych przykładów jako te klasy. Precyza jest stosunkowo wyższa, co oznacza, że gdy model przypisał już jakąś etykietę tym przykładom, to zostały poprawnie zaklasyfikowane w 40% i 33%. Uznaliśmy, że w systemie klasyfikacji prawdziwości treści bardziej istotne jest poprawne zaklasyfikowanie przypadku niż wykrycie konkretnych klas. Z tego powodu te wyniki, choć niskie, to mogą być w pewien sposób zadowalające.

Wyniki dla klas FALSE, HALF-TRUE i MOSTLY-TRUE są relatywnie lepsze, ponieważ mają wartości miary F1 w przedziale [0.29, 0.36]. Liczebność przykładów w tych klasach jest wyższa niż w PANTS-FIRE i TRUE, co może sugerować, że model faworyzuje liczniejsze klasy.

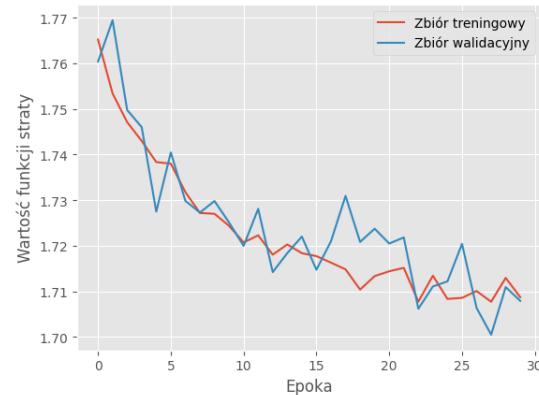
W Tabeli 5.3 zostały również umieszczone wartości miary F1 dla poszczególnych klas dla modeli LR i BiLSTM, które zostały opisane w zbiorze LIAR PLUS [3]. Twórcy zbioru LIAR PLUS

prawdopodobnie wykorzystali ważone uśrednianie miary F1, więc abyśmy mogli porównywać ich miary z naszą, to obliczono ich wartość przy pomocy uśredniania macro, które zastosowaliśmy do modelu bazowego. W oryginale F1 LR wynosi 0.23 a F1 BiLSTM wynosi 0.26. Wydaje się, że twórcy zbioru uznali, że klasy nie są na tyle zbalansowane, aby używać uśredniania macro i mogli użyć średniej ważonej, aby lepiej reprezentować udział poszczególnych klas. Patrząc na wartości miary F1 modelu bazowego i modeli ze zbioru LIAR PLUS, można zauważać, że model bazowy ma podobne miary do modelu BiLSTM i znaczaco różni się dla klas PANTS-FIRE i TRUE.

RYSUNEK 5.3: Porównanie dokładności na zbiorze treningowym i walidacyjnym podczas trenowania modelu bazowego



RYSUNEK 5.4: Porównanie wartości funkcji straty na zbiorze treningowym i walidacyjnym podczas trenowania modelu bazowego



Obserwując wykresy dokładności i wartości funkcji straty mierzonych podczas trenowania można zauważać, że na żadnym etapie model raczej nie został przetrenowany. A nawet jeśli to stało, to mechanizm zapisywania modelu z najlepszą dokładnością na zbiorze walidacyjnym chroni nas przed tym potencjalnym problemem (Rysunki 5.3 i 5.4).

5.4 Zbadanie możliwych wariantów wejścia

Po zaimplementowaniu bazowego modelu należało ustalić sposób przekazywania dodatkowych kolumn do modelu, poza kolumną *statement*. Okazuje się, że aby to zrobić, to trzeba rozróżnić rodzaje danych, jakie znajdują się w zbiorze. Niektóre dane są wartościami liczbowymi, które można bezpośrednio przekazać do modelu jako wejście, a inne są wartościami tekstowymi. Z tego powodu należy tak jak dla kolumny *statement* stworzyć odpowiednie reprezentacje kolumn tekstowych w postaci liczb, aby model mógł je efektywnie przetwarzać oraz skonkatenować z wartościami kolumn liczbowymi.

Na tym etapie uwzględniliśmy kolumny tekstowe takie jak: *statement*, *subject*, *speaker*, *job_title*, *state*, *party_affiliation*, *context*, *justification* oraz kolumny liczbowe takie jak: *barely_true_counts*, *false_counts*, *half_true_counts*, *mostly_true_counts*, a także *pants_on_fire_counts*. Przykładowe wartości uwzględnionych kolumn zostały wypisane w Tabeli 5.4.

Już we wcześniejszym etapie zauważaliśmy, że kolumna *statement* zawiera taki korpus dokumentów, których słownik jest bardzo duży, a zatem liczba kombinacji tych słów jest ogromna i dlatego zdecydowaliśmy się wykorzystać dystrybucyjną reprezentację jej treści, którą otrzymujemy dzięki zastosowaniu modelu RoBERTa. Opisaliśmy czym jest dystrybucyjna reprezentacja treści w sekcji 2.7.

W przypadku kolumny *subject*, jeden rekord może zawierać wiele tematów. Łącznie w zbiorze znajduje się 27,796 unikalnych tematów. Ich reprezentacja w postaci wektora *bag-of-words* skut-

TABELA 5.4: Przykład wartości kolumn wybranych do eksperymentu dla elementu ze zbioru treningowego

Kolumna	Przykładowa wartość
statement	Rep. Paul Ryans Medicare plan lines the pockets of the private...
subject	deficit,federal-budget,medicare,taxes
speaker	gerry-connolly
job_title	U.S. Representative
state	Virginia
party_affiliation	democrat
context	a press release.
justification	Lets review: The liberal economists dont like Ryans plan and...
barely_true_counts	-0.5556349590692453
false_counts	-0.5515403382450226
half_true_counts	-0.3670914245942747
mostly_true_counts	-0.4273303275358542
pants_on_fire_counts	-0.384859370953296

kowałyby wejściem o bardzo wysokiej liczbie wymiarów, co — zgodnie z opisany w sekcji 2.6 przekleństwem wymiarowości — mogłyby negatywnie wpływać na wydajność modelu. Z tego powodu również w tym przypadku zdecydowano się na reprezentację dystrybucyjną z wykorzystaniem modelu RoBERTa.

Podobnie kolumna *job_title*, posiada 1,360 unikalnych wartości, gdzie tokeny składające się na wartości tej kolumny zawierają się w słowniku o rozmiarze 1,290 tokenów. Z tego powodu również lepiej zakodować ją jako embedding RoBERTy.

Kolumny *context* oraz *justification*, z uwagi na swoją długość i różnorodność językową, także lepiej jest zakodować przy pomocy modelu RoBERTa.

W przypadku kolumn *state* i *party_affiliation*, liczba możliwych wartości (odpowiednio 69 i 24) jest na tyle ograniczona, że efektywniej jest przedstawić je w postaci kodowania one-hot niż za pomocą embeddingów z modelu językowego.

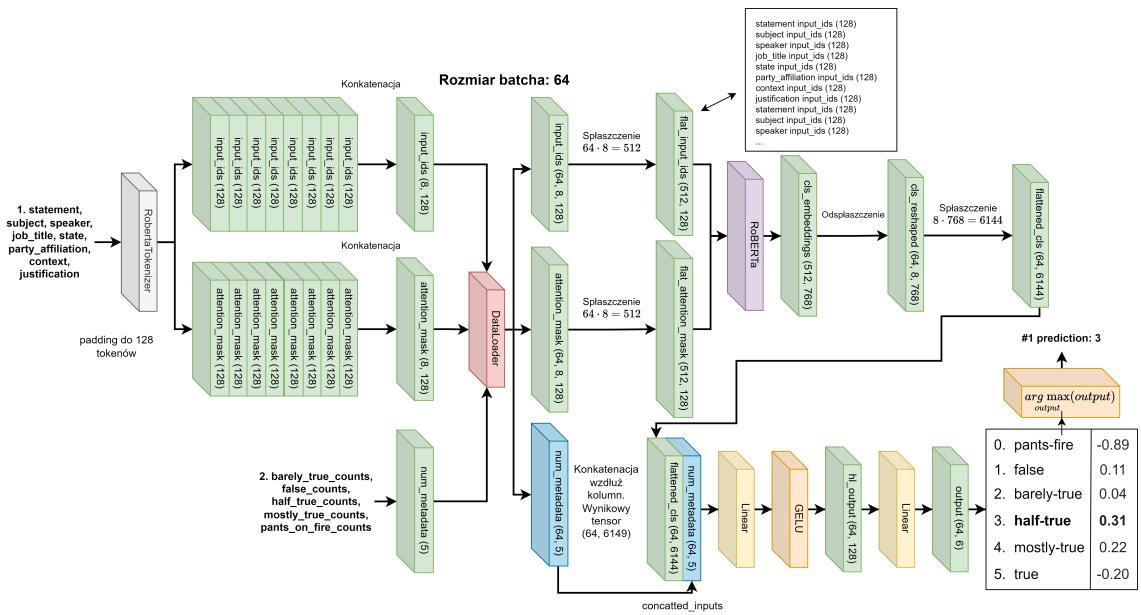
Nie było dla nas pewne, czy lepiej jest zakodować odpowiednie kolumny tekstowe przy pomocy modelu RoBERTa osobno, czy złożyć ją w jeden tekst wejściowy, z podziałem na sekcje. Dlatego zdecydowaliśmy się to sprawdzić, a także zbadaliśmy czy dotrenowanie ostatnich warstw modelu RoBERTa poprawi jakość klasyfikacji, a przez to pewnie też jakość generowanych embeddingów do naszego zadania.

5.4.1 Architektura LiarPlusMultipleRoBERTasClassifier

W pierwszym kroku zaimplementowaliśmy `LiarPlusMultipleRoBERTasClassifier`, który zamienia każdą z wymienionych powyżej kolumn tekstowych osobno na tensor tokena CLS wygenerowany przez model RoBERTa - niezależnie od trafności kodowania ich w ten sposób, co zostało omówione wcześniej (Rysunek 5.5). Model RoBERTa ma wszystkie wagi zamrożone. Wykorzystaliśmy te same parametry tokenizatora co w sekcji 5.3.

Cechy wejściowe są dostarczane przez klasę `LiarPlusDatasetSubset` lub `LiarPlusDataset`. Są to prawie identyczne klasy, z tą różnicą, że ta pierwsza ma możliwość podania rozmiaru zbioru i ziarna do generatora pseudolosowego, który wybiera losowe elementy z oryginalnego zbioru, które są potrzebne do idealnego zrównoważenia liczby przykładów w każdej z klas. Na początku eksperymentów testowaliśmy również trenowanie na mniejszym i idealnie zrównoważonym zbiorze w celu przyspieszenia trenowania, ponieważ początkowo wykorzystywaliśmy komputery lokalne, ale gdy odkryliśmy platformę Kaggle, która udostępnia lepsze akceleratory obliczeniowe, to zaczęliśmy wykonywać eksperymenty na całym zbiorze. Ostateczny eksperyment przeprowadziliśmy z użyciem

RYSUNEK 5.5: Architektura modelu LiarPlusMultipleRoBERTaClassifier z uwzględnieniem mini-batchowania przez DataLoader



klasy `LiarPlusDatasetSubset` na pełnych zbiorach.

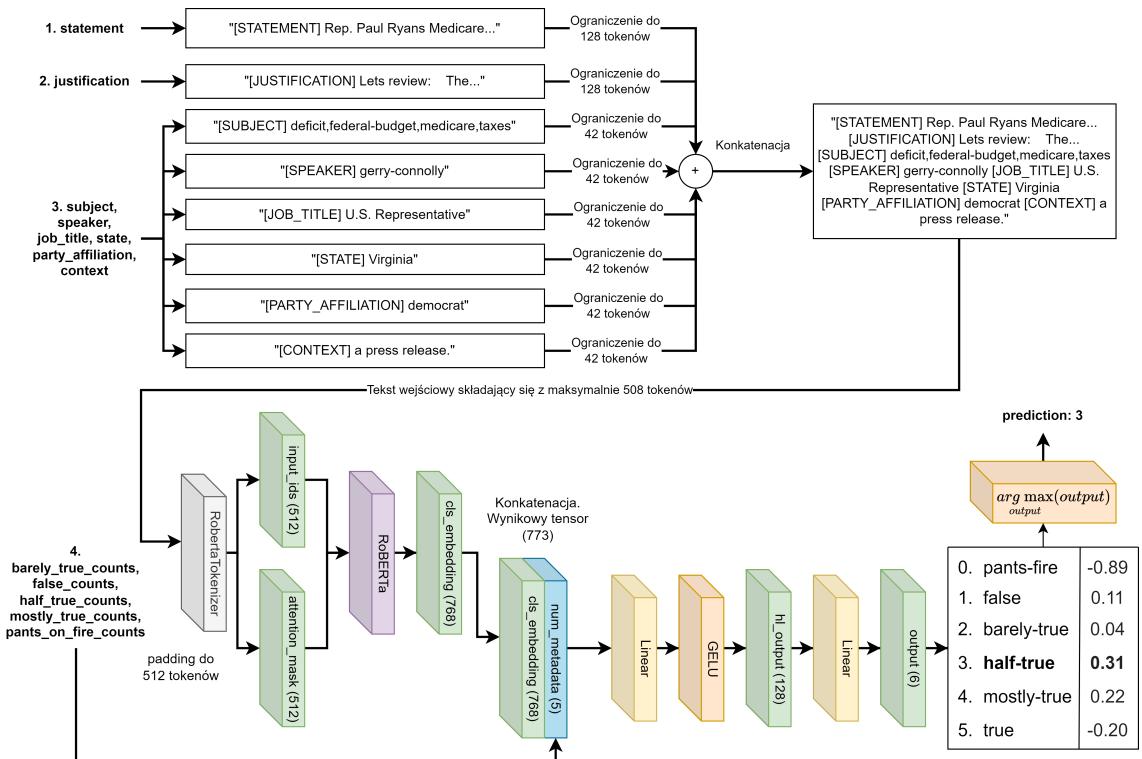
Każda z kolumn tekstowych jest osobno tokenizowana przy użyciu `RobertaTokenizer`. W wyniku tego procesu powstają tensory `input_ids` i `attention_mask`, z których każdy ma długość 128. Dzieje się to wewnątrz funkcji `__getitem__(self, index: int)` klasy `LiarPlusDatasetSubset`. Następnie te tensory, których jest po 8, są konkatenowane w rzędę, gdzie jeden rząd oznacza oryginalny tensor `input_ids` lub `attention_mask` dla danej kolumny tekstowej (np. `state`). W ten sposób powstają tensory o wymiarach 8 rzędów na 128 kolumn (np. `input_ids = torch.cat(input_ids, dim=0)`). W kolejnym kroku działania tej funkcji pobierane są cechy liczbowe i zamieniane na tensor `num_metadata`. Ponieważ jest 5 kolumn liczbowych, to tensor ma rozmiar 5 składowych.

Dalej w procesie trenowania i ewaluacji wykorzystywana jest klasa `DataLoader` z pakietu PyTorcha, która otrzymuje parametry instruujące ją aby wykonywała mieszanie przykładów (z ang. shuffle) oraz mini-batchowanie. Rozmiar batcha ustalono na 64 przykłady. Jest to istotne z perspektywy modelu, ponieważ są w nim wykonywane operacje manipulacji wymiarami tensorów wejściowych `input_ids` i `attention_mask`, tak aby zmienić je na dwa tensory wejściowe mini-batch o wymiarach po (512, 128), co jest robione po to aby przyspieszyć przetwarzanie przez model RoBERTa.

Instancja klasy `DataLoader` dostarcza tensory `input_ids`, `attention_mask` i `num_metadata`, gdzie pierwsze dwa mają wymiary (64, 8, 128), a trzeci ma wymiar (64, 5), i w każdym z nich pierwszy wymiar, równy 64, oznacza rozmiar batcha. Następnie `input_ids` i `attention_mask` są spłaszczane wzduż pierwszego i drugiego wymiaru (np. `flat_input_ids = input_ids.view(64 * 8, 128)`), w taki sposób, że każde 8 tensorów, reprezentujących wejściową cechę tekstową, staje się jednym z przykładów w batchu, który trafia jako wejście do modelu RoBERTa. 64 przykłady po 8 cech tekstowych staje się 512 przykładami wejściowymi (rzędami tensora) z perspektywy RoBERTy.

Model RoBERTa generuje tokeny o tym samym rozmiarze batcha, co wejściowy, który jest równy 512 przykładów (64 przykłady razy 8 cech tekstowych), ale o innym rozmiarze kolumn. Pobierana jest z niego wartość tokenów CLS, a następnie proces spłaszczania jest odwracany dla pierwszego wymiaru i znowu powstaje tensor z 64 przykładami, po 8 rzędów reprezentujących

RYSUNEK 5.6: Architektura modelu LiarPlusSingleRoBERTaClassifier bez uwzględnienia mini-batchowania przez DataLoader



cechy tekstowe, ale z 768 kolumnami (`cls_reshaped = cls_embeddings.view(64, 8, -1)`).

Następnie tensor tokenów CLS jest ponownie spłaszczały - tym razem na drugim i trzecim wymiarze (`flattened_cls = torch.flatten(cls_reshaped, start_dim=1)`) - tworząc tensor o wymiarach (64, 6144). Zostaje on połączony (skonkatenowany) z tensorem `num_metadata`, a wynikowy tensor trafia do warstwy ukrytej `h1`, złożonej z 128 neuronów liniowych oraz funkcji aktywacji GELU, co efektywnie wprowadza nieliniowość do modelu. Liczba przykładów w mini-batchu nadal wynosi 64, jednak liczba cech (kolumn) wejściowych do warstwy `h1` zwiększyła się po dołączeniu cech liczbowych z `num_metadata`.

Po przejściu przez warstwę ukrytą powstaje tensor o wymiarach (64, 128), który trafia do kolejnej, wyjściowej już, warstwy liniowej `fc`, która ma 6 neuronów. Powstaje tensor wynikowy `output` dla całego batcha, który ma wymiary (64, 6) i reprezentuje logity dla każdego przykładu w mini-batchu, z którego pobierane są ostateczne identyfikatory kolumn z największymi wartościami, reprezentujące wybrane przez model klasy prawdziwości przykładów.

Funkcja GELU została użyta jako funkcja aktywacji warstwy ukrytej ze względu na swoje lepsze właściwości w porównaniu do funkcji ReLU - nie powoduje problemu „umierającego ReLu”, który został opisany w podrozdziale 2.5.2, a ponadto jest stosowana również w architekturze RoBERTa do wprowadzania nieliniowości.

5.4.2 Architektura LiarPlusSingleRoBERTaClassifier

W drugim kroku zaimplementowaliśmy model `LiarPlusSingleRoBERTaClassifier` (Rysunek 5.6), który konkatenuje wszystkie cechy tekstowe w jeden większy ciąg tekstowy, w którym cechy są oddzielone tagami identyfikującymi następującą za nimi treść cechy. Tagi są zdefiniowane jako nazwa kolumny tekstowej, zapisana wielkimi literami i zamknięta z obu stron klamrami kwadratowymi, np. `[STATE]`. Pomiędzy tagami są spacje, ale pierwszy tag `[STATEMENT]` ma spację tylko

z prawej strony.

Podobnie jak w poprzednim modelu, wszystkie cechy tekstowe są uwzględnione wewnątrz wspomnianego większego ciągu znaków, który jest przeznaczony jako wejście do modelu RoBERTa, niezależnie od trafności kodowania ich w ten sposób. Wszystkie wagi modelu RoBERTa są zamrożone, tak jak w modelu **LiarPlusMultipleRoBERTasClassifier**.

Bardziej opłaca się przesłać jeden większy ciąg znaków do modelu, niż przesyłać niektóre cechy o mniejszej liczbie kombinacji osobno w formie wektorów bag-of-words – rozmiar tensora wejściowego jest mniejszy, a jednocześnie zakładamy, że RoBERTa jest w stanie wygenerować embedding, zachowujący semantycznie większość przekazanych w ten sposób informacji.

Ponieważ wszystkie cechy tekstowe są kodowane do jednego wejściowego ciągu tekstu, to całkowity limit tokenów jest tym razem ustalony na 512 tokenów, czyli maksymalny rozmiar kontekstu dla modelu RoBERTa. Aby model RoBERTa mógł efektywnie zakodować całą istotną część informacji, to przed skonstruowaniem wejściowego ciągu znaków z cech tekstowych, wszystkie te kolumny zostały ograniczone do ustalonej liczby tokenów. Bez tego niektóre cechy mogłyby być zapełnić cały maksymalny rozmiar kontekstu, co byłoby niekorzystne ze względu na to, że model **LiarPlusSingleRoBERTaClassifier** nie otrzymywałby informacji z innych kolumn. Tylko część tekstu mieszcząca się wraz ze swoim tagiem w limicie tokenów została uwzględniona do konkatenacji w większy ciąg. Ze względu na ważność kolumn *statement* i *justification*, zdecydowaliśmy się przypisać im po równo 128 tokenów z dostępnego limitu, a całą resztę rozmiaru kontekstu, czyli $512 - 2 \cdot 128 = 256$ tokenów, podzieliśmy po równo do każdej mniej istotnej cechy, czyli po $\lfloor \frac{256}{6} \rfloor = \lfloor 42\frac{1}{3} \rfloor = 42$ tokeny. Wyższa istotność kolumn *statement* i *justification* wynika z tego, że kolumna *statement* stanowi główny cel klasyfikacji, a kolumna *justification*, zawierająca średnio 72.078 słowa, została wskazana przez *Alhindi et al.* jako ta, która poprawia jakość klasyfikacji [3]. Wartość 128 tokenów jako maksymalna długość sekwencji dla tych kolumn została przyjęta na podstawie konsultacji eksperckiej z promotorem pracy, mającej na celu zapewnienie optymalnego kompromisu między jakością reprezentacji a efektywnością obliczeniową.

W dalszych krokach tensor `cls_embedding`, reprezentujący token CLS pobrany z wyniku działania modelu RoBERTa, jest konatenowany z tensorem `num_metadata` i dalsze przetwarzanie jest takie samo jak w modelu **LiarPlusMultipleRoBERTasClassifier**.

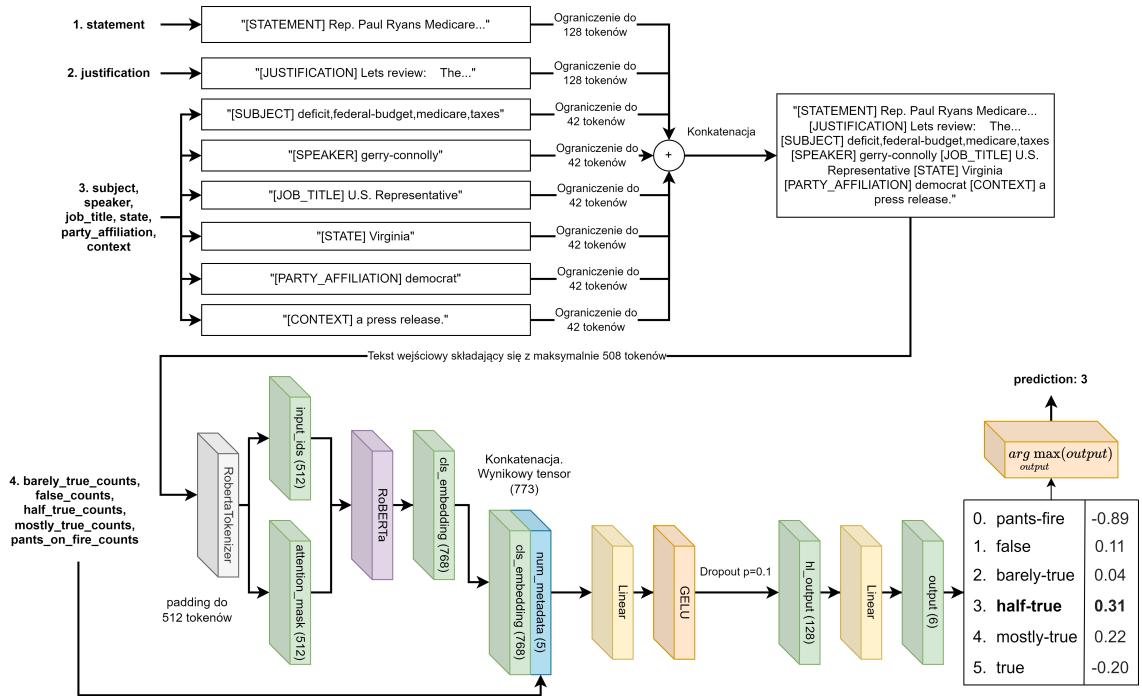
5.4.3 Architektura **LiarPlusSingleFinetunedRoBERTaClassifier**

Następnie zaimplementowaliśmy model **LiarPlusSingleFinetunedRoBERTaClassifier**, który ma prawie taką samą architekturę jak model **LiarPlusSingleRoBERTaClassifier**, ale różni się od niego tym, że pomiędzy warstwą ukrytą `h1` a warstwą wyjściową `fc` jest warstwa dropout z parametrem $p = 0.1$ (Rysunek 5.7). Dodatkowo model RoBERTa nie ma tym razem zamrożonych wag dla warstwy 12 (ostatniej) i dla warstwy poolera, co oznacza, że uczestniczą w treningu. Wartość tensora `cls_embedding` to wartość warstwy `pooler_output` modelu RoBERTa, który dostarcza pakiet `transformers` z Hugging Face. Warstwa `pooler_output` jest dodatkową warstwą selekcji tokenu CLS, którą można dotrenować, w celu otrzymywania potencjalnie lepszej reprezentacji dystrybucyjnej.

Na potrzebę trenowania tego modelu definiujemy dodatkowy parametr `encoder_lr`, który steruje hiperparametrem learning rate wszystkich parametrów (wag) modelu RoBERTa. Parametr `lr` osobno steruje hiperparametrem learning rate warstwy `h1` i `fc`.

Parametr `lr` wynosi $1e-3$, a parametr `encoder_lr` jest mniejszy i wynosi $1e-5$. Mniejsza wartość learning rate dla warstw modelu RoBERTa jest spowodowana obawą przed przeuczeniem. Podjęliśmy decyzję poprzez konsultację ekspercką z promotorem pracy, która miała na celu usta-

RYSUNEK 5.7: Architektura modelu LiarPlusSingleFinetunedRoBERTaClassifier bez uwzględnienia minibatchowania przez DataLoader. Opisuje również model LiarPlusSingleRoBERTaDropoutClassifier.



lenie takich parametrów, aby zapobiec przeuczeniu. W późniejszym etapie dowiedzieliśmy się skąd ta praktyka się wywodzi. Okazuje się, że praca „Universal Language Model Fine-tuning for Text Classification” [37] dostarcza zaleceń zgodne z tym co zrobiliśmy, które zakładają wykorzystanie mniejszych wartości learning rate dla wcześniejszych warstw, aby zapobiec przeuczeniu przy dostrajaniu modeli. Ta metoda nazywa się *discriminative fine-tuning* i zakładaając, że ostatnia warstwa ma learning rate równy $1e-3$, to można obliczyć, że learning rate ostatniej warstwy enkodera RoBERTy powinien wynosić w zaokrągleniu do siedmiu miejsc po przecinku $1e-3 \cdot (\frac{1}{2.6})^3 = 5.69e-5$, co jest blisko użytej przez nas wartości (jest w tym samym rzędzie wielkości). Wynika to ze wzoru na najlepsze wartości hiperparametru learning rate dla poprzednich warstw, który przedstawiono w omawianej pracy:

$$\eta^{l-1} = \eta^l / 2.6$$

, gdzie:

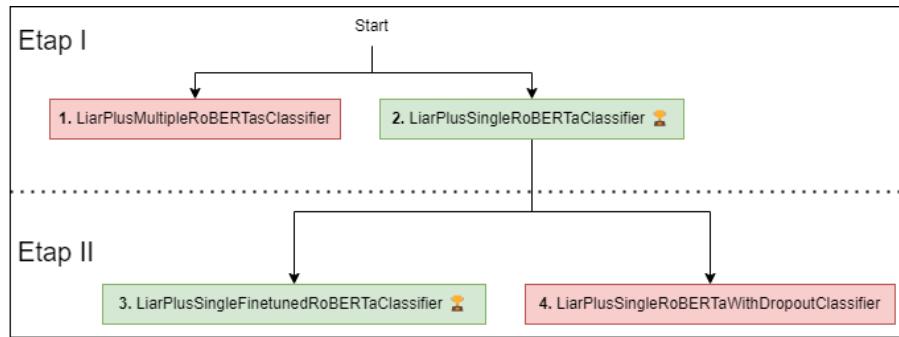
- η^l - Oznacza learning rate warstwy o numerze l .
- 2.6 - Empirycznie ustalona stała, która najlepiej zabezpiecza przed przeuczeniem.

Od ostatniej warstwy LiarPlusSingleFinetunedRoBERTaClassifier do ostatniej warstwy enkodera RoBERTy są 3 warstwy (idąc od końca): fc, hl i pooler_output - stąd stała mnożenia wynosi $(\frac{1}{2.6})^3$, bo dzielimy wartość learning rate 3 razy.

5.4.4 Architektura LiarPlusSingleRoBERTaDropoutClassifier

Po wytrenowaniu i zbadaniu wyników trzech powyższych modeli zauważliśmy, że warto sprawdzić jeszcze co się stanie, jeśli zastosujemy architekturę LiarPlusSingleRoBERTaClassifier, ale dodamy warstwę dropout, tak jak w modelu LiarPlusSingleFinetunedRoBERTaClassifier

RYSUNEK 5.8: Przebieg analizy eksperymentów z możliwymi wariantami wejścia



(Rysunek 5.7). Zaimplementowaliśmy model **LiarPlusSingleRoBERTaDropoutClassifier**, który różni się od wspomnianego modelu właśnie tym. Schemat modelu jest taki sam, ale różnią się metodą treningu. Wciąż, wszystkie wagi modelu RoBERTa są zamrożone i nie uczestniczą w treningu.

5.4.5 Porównanie i wybór najlepszej architektury

Wytrenowaliśmy wszystkie modele przy użyciu takich samych parametrów co w sekcji 5.3, no chyba że jasno zdefiniowano, że użyto innych parametrów i wypisano jakie, jak w przypadku modelu **LiarPlusSingleFinetunedRoBERTaClassifier**.

TABELA 5.5: Miary na zbiorze walidacyjnym wytrenowanych modeli dotyczących eksperymentów na temat najlepszego sposobu przetwarzania tekstowych cech wejściowych

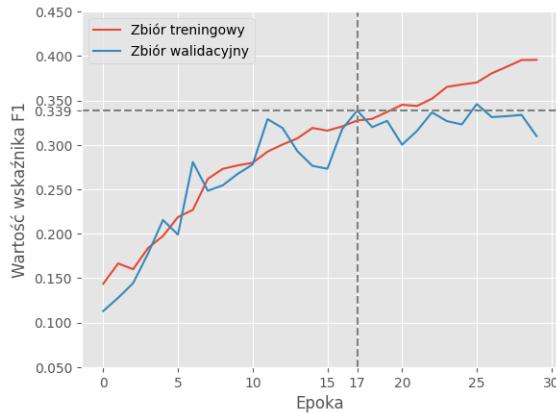
Nazwa modelu	Learning rate	Zakrąglony czas treningu [h]	Liczba wykonanych epok	Dokładność	Wartość funkcji straty	F1	Precyzja	Pelność
LiarPlusSingleFinetunedRoBERTaClassifier	1e-3 - hl, fc; 1e-5 - RoBERTa	2.8	30	0.34	1.60	0.35	0.37	0.34
LiarPlusSingleRoBERTaWithDropoutClassifier	1e-3	1.5	18	0.27	1.72	0.20	0.26	0.24
LiarPlusSingleRoBERTaClassifier	1e-3	2.6	30	0.27	1.70	0.18	0.25	0.24
LiarPlusMultipleRoBERTasClassifier	1e-3	3.2	28	0.23	1.74	0.11	0.12	0.19

Na początku wytrenowano model **LiarPlusMultipleRoBERTasClassifier**, a następnie model **LiarPlusSingleRoBERTaClassifier**. Zauważliśmy, że model, który przetwarza wszystkie cechy tekstowe w postaci jednego wektora zanurzeń, ma lepsze wszystkie wyniki od tego, który tworzy osobne wektory zanurzeń. Nie uwzględniamy liczby wykonanych epok przy podejmowaniu decyzji, ponieważ ona mówi jedynie, kiedy model przestał się uczyć, i potencjalnie też kiedy model znalazł najlepsze parametry. Uczyliśmy modele wielokrotnie i każdy model wykonywał do 30 epok. Żaden nie wyróżniał się na tym polu.

Ze względu na to, że to model **LiarPlusSingleRoBERTaClassifier** osiągnął w pierwszej fazie lepsze wyniki niż **LiarPlusMultipleRoBERTasClassifier**, to na jego podstawie rozwinęliśmy architekturę o fine-tuning modelu RoBERTa. Kolejność eksperymentów jest pokazana na Rysunku 5.8. Przy każdym modelu jest numer operacji. Symbol pucharu oraz zielony kolor tła tekstu informuje o tym, który model został wybrany do dalszego etapu. Czerwony kolor tła tekstu oznacza gorszy wariant.

Na Tabeli 5.5 można zauważyć pomiary wytrenowanych modeli na zbiorze walidacyjnym. Widac, że model **LiarPlusSingleFinetunedRoBERTaClassifier** osiągnął najlepsze wyniki, które zdecydowanie przewyższają wyniki innych modeli. Najgorzej wypadł model wykorzystujący wiele reprezentacji tekstu - **LiarPlusMultipleRoBERTasClassifier**. Po środku rankingu jest model **LiarPlusSingleRoBERTaClassifier** z dropoutem oraz wersja bez dropoutu. Wersja z dropoutem

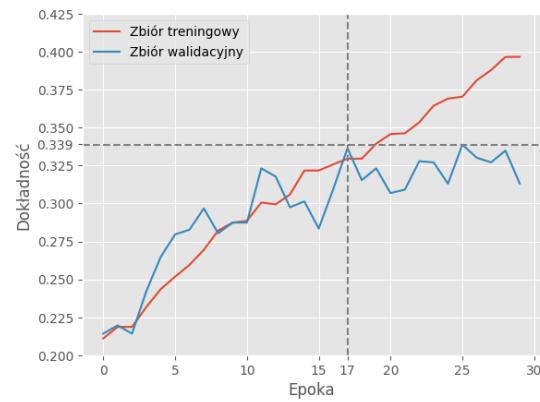
RYSUNEK 5.11: Porównanie wartości miary F1 modelu LiarPlusSingleFinetunedRoBERTaClassifier podczas trenowania



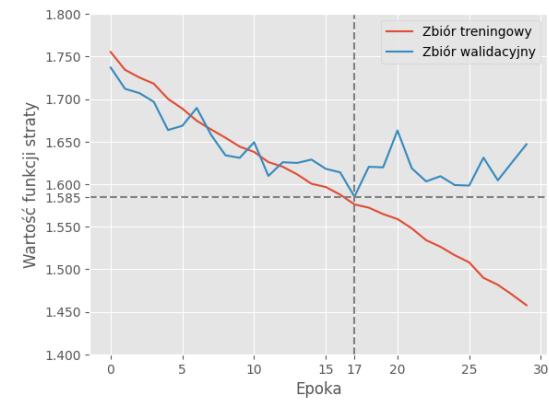
została zaimplementowana na końcu, w celu sprawdzenia, jak bardzo dropout wpływa na poprawę wyniku.

Spoglądając na wykresy dokładności, wartości funkcji straty oraz miary F1 dla zbioru treningowego i walidacyjnego, można zauważać oznaki przeuczenia (Rysunki 5.9, 5.10, 5.11). Model osiągnął niemal najlepszą dokładność na zbiorze walidacyjnym już w 17. epoce, a w kolejnych epokach wartości miar nie ulegały znaczącej poprawie – wykresy sugerują ich stabilizację wokół poziomów z 17. epoki. Mechanizm wczesnego zatrzymania (*early stopping*) nie został aktywowany, ponieważ w 26. epoce model osiągnął nieznacznie wyższą dokładność niż wcześniej, co spowodowało zapisanie tej wersji modelu jako najlepszej. Można przypuszczać, że zastosowanie mniejszej wartości parametru *patience* (np. 6 zamiast 10) mogłoby skrócić czas trenowania, a nawet poprawić zdolność generalizacji. Mimo wszystko, model z 26. epoki charakteryzuje się lepszą wartością miary F1 niż w 17. epoce, co może świadczyć o lepszym dopasowaniu do danych oraz potencjalnie wyższej jakości klasyfikacji.

RYSUNEK 5.9: Porównanie dokładności modelu LiarPlusSingleFinetunedRoBERTaClassifier podczas trenowania

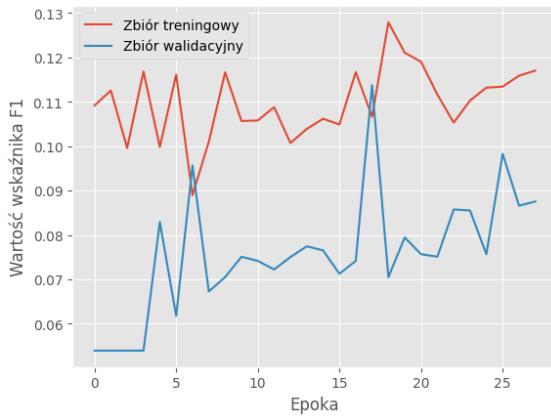


RYSUNEK 5.10: Porównanie wartości funkcji straty modelu LiarPlusSingleFinetunedRoBERTaClassifier podczas trenowania



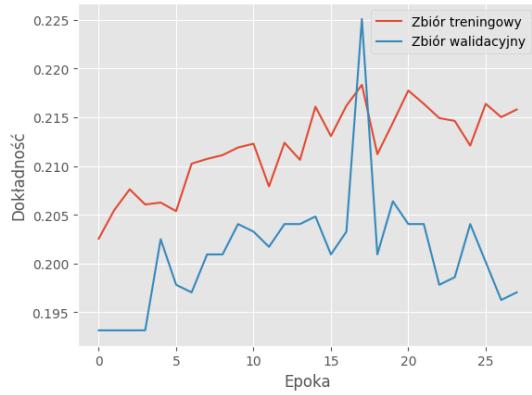
Przechodząc na model, który osiągnął najgorsze wyniki w eksperymencie, można zauważać na wykresach dokładności, wartości funkcji straty oraz miary F1, że dokładność na zbiorze walidacyjnym podąża po podobnej krzywej co dokładność na zbiorze treningowym, ale jest od niej prawie zawsze oddalona, z wyjątkiem epoki 17, w której dokładność na zbiorze walidacyjnym jest wyższa niż na zbiorze treningowym (Rysunki 5.12, 5.13, 5.14). Może się wydawać, że model ma duże trud-

RYSUNEK 5.14: Porównanie wartości miary F1 modelu LiarPlusMultipleRoBERTasClassifier podczas trenowania

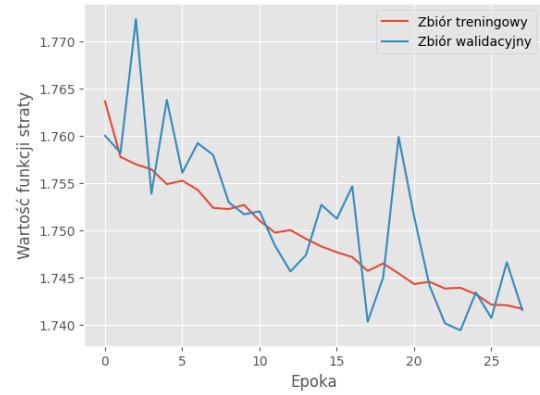


ności z osiągnięciem zbliżonej generalizacji na zbiorze walidacyjnym, co na zbiorze treningowym. Co ciekawe wartość funkcji straty na obu zbiorach stale malała w każdym etapie, ale cały spadek jest mniejszy niż w przypadku modelu **LiarPlusSingleFinetunedRoBERTaClassifier**. Być może model osiągnąłby lepsze wyniki, gdyby pozwolić mu się dłużej trenować, ale to i tak by dyskwalifikowało to podejście ze względu na koszt czasowy. Trenowanie tego modelu zabrało już 3.2h, a kolejne zwiększanie tego czasu nie gwarantuje prześcignięcia wyników najlepszego modelu - czyli **LiarPlusSingleFinetunedRoBERTaClassifier**. Wartość miary F1 zachowuje podobną zależność co dokładność.

RYSUNEK 5.12: Porównanie dokładności modelu LiarPlusMultipleRoBERTasClassifier podczas trenowania

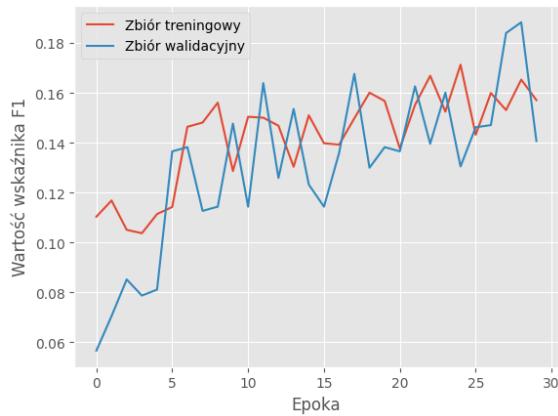


RYSUNEK 5.13: Porównanie wartości funkcji straty modelu LiarPlusMultipleRoBERTasClassifier podczas trenowania



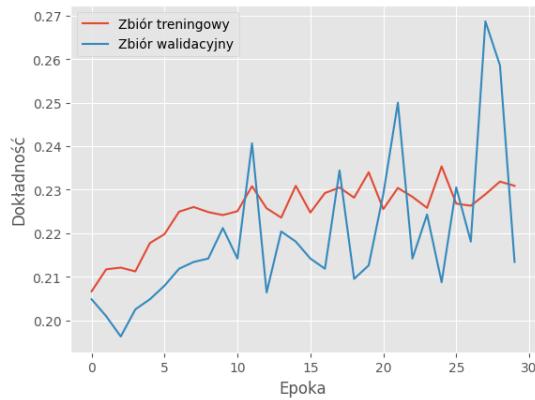
Modele **LiarPlusSingleRoBERTaClassifier** oraz jego wersja z dropoutem nie wykazują niezwykłych wzorców w wykresach (Rysunki 5.15, 5.16, 5.16, 5.18, 5.19, 5.20). Dokładność wersji modelu bez dropoutu na zbiorze walidacyjnym zbliża się do dokładności na zbiorze treningowym. Podobnie wersja z dropoutem, chociaż na niej można zauważać większe różnice w wartościach, gdy dokładność na zbiorze walidacyjnym przewyższa tą na zbiorze treningowym. W obu przypadkach wartość funkcji straty maleje. Można zauważać, że od epoki 10. poprawy w metrykach wynikają z nieprzewidywalnych skoków, a nie stopniowej zmiany. Patrząc na wykresy modelu z dropoutem, można wywnioskować, że zastosowanie dropoutu, nawet bez fine-tuningu, delikatnie poprawia dokładność (więcej skoków, z wyższymi wartościami niż bez dropoutu) oraz wartość miary F1 (mniej skoków, ale mają wyższe wartości niż bez dropoutu), ale za to całkowity spadek wartości funkcji straty jest mniejszy niż dla modelu **LiarPlusSingleRoBERTaClassifier**, bo dla wersji bez dro-

RYSUNEK 5.17: Porównanie wartości miary F1 modelu LiarPlusSingleRoBERTaClassifier podczas trenowania

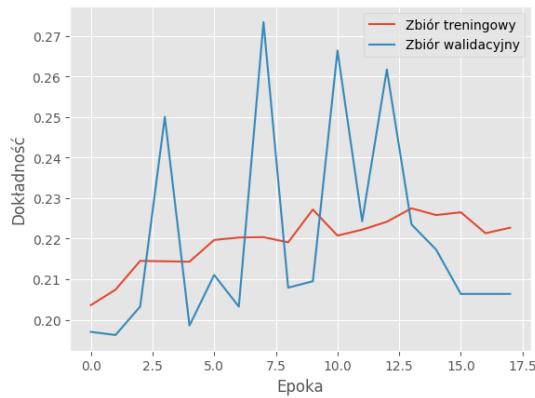


poutu wynosi w zaokrągleniu do dwóch miejsc po przecinku 0.06, a dla wersji z dropoutem wynosi w zaokrągleniu do dwóch miejsc po przecinku 0.04, więc różnica to 0.02.

RYSUNEK 5.15: Porównanie dokładności modelu LiarPlusSingleRoBERTaClassifier podczas trenowania



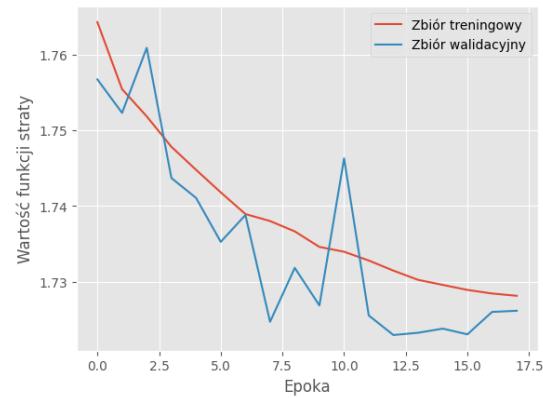
RYSUNEK 5.18: Porównanie dokładności modelu LiarPlusSingleRoBERTaWithDropoutClassifier podczas trenowania



RYSUNEK 5.16: Porównanie wartości funkcji straty modelu LiarPlusSingleRoBERTaClassifier podczas trenowania



RYSUNEK 5.19: Porównanie wartości funkcji straty modelu LiarPlusSingleRoBERTaWithDropoutClassifier podczas trenowania



W celu zbadania różnic w klasyfikacji poszczególnych klas między najlepszym a najgorszym modelem, zestawiono ich miary precyzji, pełności i wartości miary F1 dla najlepszych wersji

RYSUNEK 5.20: Porównanie wartości miary F1 modelu LiarPlusSingleRoBERTaWithDropoutClassifier podczas treningu

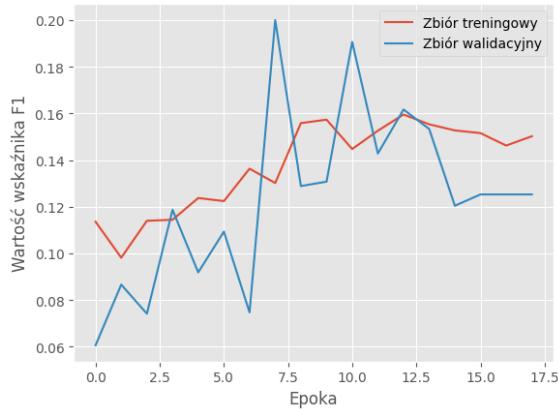


TABELA 5.6: Porównanie precyzji, pełności i wartości miary F1 na zbiorze walidacyjnym dla najlepszych wytrenowanych wersji modeli LiarPlusSingleFinetunedRoBERTaClassifier oraz LiarPlusMultipleRoBERTasClassifier (wszystkie miary są uśrednione macro). Wartości 0 są pogrubione, aby uwypuklić różnice.

Klasa	Rozmiar klasy	LiarPlusSingleFinetuned RoBERTaClassifier			LiarPlusMultiple RoBERTasClassifier		
		Precyzja	Pełność	F1	Precyzja	Pełność	F1
pants-fire	116	0.62	0.37	0.46	0.00	0.00	0.00
false	263	0.34	0.30	0.32	0.29	0.21	0.24
barely-true	237	0.34	0.30	0.32	0.00	0.00	0.00
half-true	248	0.30	0.40	0.34	0.20	0.06	0.10
mostly-true	251	0.36	0.41	0.38	0.22	0.87	0.35
true	169	0.25	0.24	0.25	0.00	0.00	0.00
suma/średnia	1284	0.37	0.34	0.35	0.12	0.19	0.11

tych modeli, zmierzone na zbiorze walidacyjnym w Tabeli 5.6. Na wykresach można zauważyć, że gorszy model nie wykrywa klas PANTS-FIRE, BARELY-TRUE i TRUE, co objawia się zerowymi wartościami precyzji, pełności i F1-score, podczas gdy wszystkie wartości miary F1 dla najlepszego modelu są niezerowe. Nawet klasa pants-fire jest wykrywana w porównaniu z modelem bazowym (Tabela 5.3), a także precyzja tej klasy jest najwyższa z pośród wszystkich dla modelu LiarPlusSingleFinetunedRoBERTaClassifier. Podczas gdy najlepszy model wykazuje bardziej zrównoważony rozkład wartości współczynnika F1, to F1 najgorszego modelu jest niezerowa jedynie dla klas FALSE, HALF-TRUE i MOSTLY-TRUE, a pełność klasy MOSTLY-TRUE osiąga 0.87, co może świadczyć o tym, że model skupia się na wykrywaniu głównie tej klasy, trochę mniej klasy FALSE i minimalnie klasy HALF-TRUE, ignorując inne klasy, co negatywnie wpływa na jego ogólną skuteczność. Wartości miary F1 są wyższe w modelu dotrenowanej RoBERTy niż w modelu bazowym dla klas PANTS-FIRE, BARELY-TRUE, HALF-TRUE, MOSTLY-TRUE, i TRUE, co wskazuje, na poprawę skuteczności klasyfikacji na pięciu z sześciu pól.

Biorąc pod uwagę wyniki wartości metryk, przedstawionych na Tabeli 5.5 oraz Tabeli 5.6, zdecydowaliśmy wykorzystywać architekturę LiarPlusSingleFinetunedRoBERTaClassifier do dalszych prac. Architektura LiarPlusMultipleRoBERTasClassifier wymaga zbyt wiele czasu do trenowania i nie dostarcza zadowalających wyników.

Rozdział 6

Rozszerzenie modelu o generację treści

Dodatkowym zadaniem, które zostało zrealizowane to wygenerowanie artykułów na podstawie klasyfikowanych wypowiedzi. Generowanie artykułów ma na celu poprawę wyników klasyfikatora, poprzez dodanie kontekstu i większej ilości faktów na temat klasyfikowanej wypowiedzi. Oparliśmy się na założeniu, że duże modele językowe mają ogólną wiedzę o świecie, którą możemy wykorzystać do wzbogacenia danych, na których będzie działał klasyfikator. Wygenerowane artykuły zostały umieszczone w zbiorze danych jako dodatkowa kolumna.

6.1 Wybór modelu LLM do generowania treści

Podczas wyboru modelu do generowania treści głównymi kryteriami była jakość danych generowanych przez model, jak i jego wszechstronność. Niepożądana byłaby sytuacja, w której model dawałby różnej jakości wyniki w zależności od tematu artykułu. Ograniczeniem były skończone zasoby obliczeniowe. Komputer, który został wykorzystany do generowania artykułów jest komputerem stacjonarnym, wykorzystywanym prywatnie. Najważniejsze parametry tego komputera, to:

- Procesor - Intel i5-11600K.
- Rozmiar pamięci RAM - 16GB.
- Karta graficzna - NVIDIA GeForce RTX 3060.

Wybrany został model Gemma 2 2B IT, który rozwijany jest przez firmę Google [72]. Gemma 2 jest dostępny w wersjach z 2, 9 i 27 miliardami parametrów. W pracy został wykorzystany model z 2 miliardami parametrów. Jest to model oparty na tych samych rozwiązańach co modele z rodziny Gemini. Wykorzystany w pracy wariant został wytrenowany na danych o zróżnicowanym pochodzeniu, których łączna długość to 2 biliony tokenów. Głównym językiem w danych do trenowania, był język angielski. Przy trenowaniu modelu wykorzystywany był finetuning z nakierowaniem na wykonywanie instrukcji.

6.2 Proces generowania treści

Kod realizujący zadanie generowania treści był zrealizowany w takiej samej architekturze jak skrypty dodające dodatkowe metadane do zbioru danych. Dostęp do modelu był możliwy poprzez platformę huggingface. Dodatkowo wykorzystana została paczka pytorch.

Prompt wykorzystany do generowania artykułu musiał być zmieniany ze względu na to, że czasami model odmawiał wygenerowania artykułu, jeżeli wypowiedź była nieprawdziwa lub zawierała mowę nienawiści. Finalna wersja promptu, dzięki któremu model ani razu nie odmówił wygenerowania artykułu:

```
[INST] Imagine that you are a journalist working in a newspaper. Write an article for the following subject: [statement]. Please write it as one continuous block of text, no formatting, no captioning, no headings. [/INST]
```

Przy wywołaniu kodu w miejsce *[statement]* podstawiana była klasyfikowana wypowiedź, na podstawie której generowany miał być artykuł. Dodatkowe polecenia, które proszą model o konkretny styl są dodane, ponieważ bez nich model z pewną dowolnością umieszczał w tekście akapy, nagłówki i inne ozdobne formy spotykane w artykułach prasowych. Znaczniki *[INST]* i *[/INST]* zostały wykorzystane w celu kontrolowania gdzie zaczyna się i kończy prompt. Różnymi parametrami (*temperature=0.9*, *top_p=0.95*, *do_sample=True*) został osiągnięty efekt tego, że model nie wybierał za każdym razem najbardziej prawdopodobnej kontynuacji, tylko miał pewną losowość w generowaniu wyniku. Maksymalna długość wygenerowanego artykułu została ustawiona na 256 tokenów.

Następnym zadaniem była optymalizacja procesu generowania wszystkich artykułów, ponieważ po wstępnej konfiguracji model pracował z szybkością 45 sekund na jeden artykuł. W celu poprawy wydajności zastosowano kwantyzację modelu, która zredukowała precyzję przechowywanych wag modelu w pamięci z 16 do 4 bitów. Dodatkowo zastosowano przetwarzanie batchowe z wielkością batcha $n=100$ promptów. Techniki te pozwoliły na przyspieszenie pracy modelu do poziomu generowania jednego artykułu w 0.72 sekundy.

Wygenerowane artykuły były jeszcze dodatkowo przetwarzanie po to, żeby usunąć z nich zbędne białe znaki i znaki nowej linii. Tak przygotowane artykuły zostały zapisane w dodatkowej kolumnie, dołączonej do finalnego zbioru danych.

6.3 Integracja generowanej treści z klasyfikatorem

Wygenerowane artykuły integrowaliśmy z klasyfikatorem (architektura klasyfikatora - rysunek 5.7) dodając do tekstowych danych wejściowych artykuł ograniczony do 128 tokenów zaraz za kolumną *justification*. Długość wejść pochodzących z kolumn *subject*, *speaker*, *job_title*, *state*, *party_affiliation* i *context* zmniejszaliśmy dynamicznie w zależności od architektury, aby zmieścić jak najwięcej tokenów w limicie wynoszącym 512.

Rozdział 7

Eksperymenty i wyniki

7.1 Architektura rozwiązań i ich motywacje

Bazując na klasyfikatorze LiarPlusSingleFinetunedRoBERTaClassifier, którego architektura została opisana na rysunku 5.7 przeprowadziliśmy eksperymenty w następujących architekturach:

- SM - Wykorzystując *statement* i kolumny z metadanymi.
- SMJ - Wykorzystując *statement*, kolumny z metadanymi i *justification*.
- SMA - Wykorzystując *statement*, kolumny z metadanymi i wygenerowane artykuły.
- SMJA - Wykorzystując *statement*, kolumny z metadanymi, *justification* i wygenerowane artykuły.

Dodatkowo dla każdej architektury przeprowadziliśmy dodatkowy eksperiment ze zmodyfikowanym klasyfikatorem, który wykorzystywał one-hot wektory do kodowania danych kategorycznych. Dla architektur SMJ i SMJA z wykorzystaniem one-hot wektorów wykonaliśmy jeszcze jeden eksperiment, w którym uruchomiliśmy klasyfikator ze zwiększym parametrem *patience* (10, gdzie dla pozostałych eksperymentów używaną wartością było 6). Zdecydowaliśmy się zmodyfikować klasyfikator o reprezentację kolumn kategorycznych jako one-hot wektorów, ponieważ pozwoliło nam to przekazywać więcej tokenów do RoBERTy.

7.2 Porównanie klasyfikatorów

Dla eksperymentów, które nie wykorzystywały one-hot wektorów (Tabela 7.1) najlepszy we wszystkich miarach jakości okazał się klasyfikator pracujący w architekturze SMJ (dokładność 0.32, F1 0.33, precyza 0.34, pełność 0.32). SMJA wypadł najgorzej, szczególnie w F1 (0.26) i pełności (0.27), co może sugerować, że dodanie komponentu A do wariantu SMJ pogarsza uogólnienie modelu. SMA lekko traci w precyzyji (0.30) i pełności (0.29), względem SM. Można z tego wywnioskować, że komponent A nie wnosi wartości dodanej. Wartość funkcji straty jest zbliżona do siebie (1.62 dla wszystkich wariantów oprócz SMA, które ma 1.65), ale nie oddaje to różnic jakościowych między modelami. Na przykład SMJ i SMJA mają identyczną strategię, ale różnią się istotnie w F1 (0.33 do 0.26). To pokazuje, że wartość funkcji straty nie zawsze koreluje z faktyczną jakością predykcji w zadaniu wieloklasowym. Może być tak, że SMJA częściej popełnia poważniejsze błędy, które mają duży wpływ na wynik F1 (np. rzadkie klasy), podczas gdy model SMJ, mimo tej samej wartości funkcji straty popełnia błędy w mniej znaczący sposób dla F1.

TABELA 7.1: Wyniki eksperymentów bez zastosowania wektorów one-hot - miary jakości na zbiorze testowym

	SM	SMJ	SMA	SMJA
Dokładność	0.29	0.32	0.29	0.29
Funkcja straty	1.62	1.62	1.65	1.62
F1	0.29	0.33	0.29	0.26
Precyzja	0.31	0.34	0.30	0.31
Pełność	0.30	0.32	0.29	0.27

Dla eksperymentów, które wykorzystywały one-hot wektory (Tabela 7.2) model SMJ (dokładność 0.31, F1 0.30, precyzja 0.33, pełność 0.29) także okazał się najlepszy, a SMJA najgorszy (F1 0.26). W przeciwieństwie do eksperymentów, które nie wykorzystywały one-hot wektorów model SMA był delikatnie lepszy niż SM (F1 0.28 do 0.27 i pełność 0.29 do 0.26), jednak precyzję nadal SM ma większą (0.33 do 0.31). Porównując wyniki eksperymentów można zauważyc, że użycie one-hot wektorów w naszym zadaniu pogorszyło wyniki modeli, które nie wykorzystują komponentu A. Dla SM pogorszyła się dokładność ($0.29 \rightarrow 0.28$), F1 ($0.29 \rightarrow 0.27$), wartość funkcji straty ($1.62 \rightarrow 1.63$) i pełność ($0.30 \rightarrow 0.26$); za to poprawiła się precyzja ($0.31 \rightarrow 0.33$). Dla SMJ pogorszyły się wszystkie metryki: dokładność ($0.32 \rightarrow 0.31$), wartość funkcji straty ($1.62 \rightarrow 1.63$), F1 ($0.33 \rightarrow 0.30$), precyzja ($0.34 \rightarrow 0.33$) i pełność ($0.32 \rightarrow 0.29$). Zastosowanie one-hot wektorów w mniejszościennym sposobie wpłynęło na modele wykorzystujące artykuły. Dla SMJA pogorszyła się tylko wartość funkcji straty ($1.62 \rightarrow 1.63$). Poprawiła się pełność ($0.27 \rightarrow 0.28$), a reszta metryk pozostała na takim samym poziomie. Dla SMA poprawiła się wartość funkcji straty ($1.65 \rightarrow 1.63$) i precyzja ($0.30 \rightarrow 0.31$), za to F1 się pogorszyło ($0.29 \rightarrow 0.28$), a pozostałe metryki pozostały bez zmian. Te wyniki sugerują, że być może gdy przekazuje się więcej tokenów do RoBERTy, które pochodzą z kolumn zawierających dane tekstowe, to zwiększa się wartość komponentu A dla jakości klasyfikacji.

TABELA 7.2: Wyniki eksperymentów z zastosowaniem wektorów one-hot - miary jakości na zbiorze testowym

	SM	SMJ	SMA	SMJA
Dokładność	0.28	0.31	0.29	0.29
Funkcja straty	1.63	1.63	1.63	1.63
F1	0.27	0.30	0.28	0.26
Precyzja	0.33	0.33	0.31	0.31
Pełność	0.26	0.29	0.29	0.28

Dla eksperymentów, które wykorzystywały one-hot wektory i parametr `patience` ustawiony na 10 (Tabela 7.3) model SMJA okazał się lepszy od SMJ we wszystkich metrykach poza wartością funkcji straty. Dla modelu działającego w architekturze SMJ lepsze metryki zostały osiągnięte bez wydłużonego parametru `patience`, co może oznaczać, że model bez komponentu A nie potrzebuje wydłużonego czasu na trening. Jednak wyniki modelu SMJA są lepsze ze zwiększoną parametrem `patience`. Poprawiła się dokładność ($0.29 \rightarrow 0.30$), F1 ($0.26 \rightarrow 0.30$) i pełność ($0.28 \rightarrow 0.30$). Precyzja pozostała taka sama (0.31), a wartość funkcji straty się pogorszyła ($1.63 \rightarrow 1.65$). Oznacza to, że model ten poprawia się wraz z wydłużeniem czasu treningu.

TABELA 7.3: Wyniki eksperymentów z zastosowaniem wektorów one-hot i parametrem `patience=10` - miary jakości na zbiorze testowym

	SMJ	SMJA
Dokładność	0.29	0.30
Funkcja straty	1.62	1.65
F1	0.29	0.30
Precyzja	0.31	0.31
Pełność	0.28	0.30

Losowy klasyfikator w zadaniu sześcioklasowym miałby dokładność około 0.167, a wszystkie modele osiągają prawie dwukrotność tego wyniku. Można z tego wywnioskować, że modele faktycznie nauczyły się rozwiązywać zadanie. Dodatkowo udało nam się zbliżyć do wartości z artykułu LIAR PLUS [3], gdzie najlepszym wynikiem miary F1 na zbiorze testowym było 0.37. W naszym przypadku było to 0.33 dla modelu SMJ.

7.3 Analiza wpływu generowanych treści na dokładność klasyfikacji

Chcąc aby ostatecznie wykorzystany model był niezależny od **konkretnych**, zewnętrznych źródeł wiedzy, takich jak PolitiFact, zdecydowaliśmy, że nie będziemy w dalszym etapie implementować pobierania uzasadnień do kolumny *justification* dla nowych, niezaobserwowanych przykładów. Z tego powodu ustaliliśmy, że wynikowym modelem naszej pracy będzie model SMA. Jednak ponieważ model SMA wykorzystujący wektory one-hot ma wyższą precyzję niż ten sam model nie wykorzystujący wektorów one-hot, to podjeliśmy decyzję, aby dalej analizować już tylko modele z kodowaniem dodatkowych metadanych w postaci skonkatenowanego wektora one-hot.

Po wytrenowaniu modeli oraz przeprowadzeniu analizy wydajności, na etapie implementacji aplikacji demonstracyjnej zauważylismy, że wektor one-hot reprezentujący kolumnę *gibberish* ma mniejszy rozmiar, niż powinien. Różnica wynosi jedną składową. Jest tak ponieważ żaden przykład nie ma przypisanej etykiety NOISE, i z tego powodu nie spostrzegliśmy dodatkowej możliwej wartości podczas ustalania rozmiaru wektora one-hot kolumny *gibberish*. Cały skonkatenowany wektor ma zatem 22 składowe zamiast 23.

7.3.1 Metodologia analizy

W celu analizy potencjalnych przyczyn otrzymania lepszych wyników przez klasyfikator SM w stosunku do klasyfikatora SMA wykonaliśmy na **zbiorze testowym** następujące operacje:

- Zaklasyfikowaliśmy każdy przykład przy pomocy klasyfikatora SM oraz klasyfikatora SMA i otrzymaliśmy następujące kolumny:
 - SM_pred* - Identyfikator klasy przypisany przykładowi na podstawie wyniku z klasyfikatora SM.
 - SMA_pred* - Identyfikator klasy przypisany przykładowi na podstawie wyniku z klasyfikatora SMA.
 - SM_prob* - Wyjściowy wynik klasyfikatora SM, przetworzony przez funkcję softmax, dla danego przykładu, który jest wektorem reprezentującym prawdopodobieństwa przynależności do danej klasy.
 - SMA_prob* - Wyjściowy wynik klasyfikatora SMA, przetworzony przez funkcję softmax, dla danego przykładu, który jest wektorem reprezentującym prawdopodobieństwa przynależności do danej klasy.

- *label_num* - Identyfikator właściwej klasy dla danego przykładu.
- *SM_highest_prob* - Najwyższa wartość z wektora *SM_prob*.
- *SMA_highest_prob* - Najwyższa wartość z wektora *SMA_prob*.

Możliwe wartości *SM_pred* i *SMA_pred* odpowiadają liczbom w słowniku **LABEL_MAPPING**, który został opisany w rozdziale 5.3.1.

2. Na potrzeby ułatwienia porównania poziomu zdecydowania w podejmowaniu decyzji przez klasyfikator podczas dalszej szczegółowej analizy konkretnych przykładów, obliczyliśmy entropię na wartościach kolumn *SM_prob* i *SMA_prob*. Inspirując się ideą pomiaru jakości podziału zbioru w algorytmie ID3, zauważaliśmy, że można wykorzystać wzór na entropię do zmierzenia „zdecydowania” wektorów wyjściowych analizowanych modeli. Okazuje się, że podobna miara jest znana w literaturze jako jedna z miar „pewności” klasyfikatora (z ang. *confidence*) [81, 65], choć klasycznie przez „confidence” rozumie się maksymalną wartość rozkładu prawdopodobieństwa. Na nasze potrzeby zdefiniowaliśmy tę miarę jako „zdecydowanie”, wyrażone wzorem $Z(\vec{p}) = 1 - \frac{H(\vec{p})}{\log_2(6)}$ (odwrotność znormalizowanej entropii), co oznacza, że klasyfikator przyjmuje maksymalną wartość zdecydowania $Z(\vec{p}) = 1$, gdy $H(\vec{p}) = 0$, a najniższą $Z(\vec{p}) = 0$, gdy $H(\vec{p}) = \log_2(6)$.
3. Dodaliśmy kolumnę *sort_key* zdefiniowaną jako iloczyn kolumny *SM_highest_prob* oraz kolumny *SMA_highest_prob*, aby móc później posortować przykłady malejąco tak, żeby przykłady z najwyższymi prawdopodobieństwami **dla obu klasyfikatorów jednocześnie** były na początku listy, dla ułatwienia późniejszego wyboru. Założyliśmy, że najbardziej istotne do analizy będą te przykłady, przy których klasyfikatory były najbardziej „pewne” swoich decyzji (czyli ich iloczyn największych prawdopodobieństw był najwyższy). Zdefiniowaliśmy „pewność” klasyfikatora jako wartość maksymalnego elementu wektora wynikowego - $\max(\vec{p})$. Ta miara również jest powszechnie znana jako wariant miary „confidence” [81].
4. Wybraliśmy te przykłady, dla których wartość kolumny *SM_pred* **jest równa** wartości kolumny *label_num* i wartość *SMA_pred* **nie jest równa** *label_num* (czyli wybraliśmy przykłady, gdzie klasyfikator SM się nie mylił, a SMA tak). Posortowaliśmy je malejąco po wartościach kolumny *sort_key* i zapisaliśmy 30 przykładów do dalszej analizy.
5. Wybraliśmy te przykłady, dla których wartość kolumny *SM_pred* **nie jest równa** wartości kolumny *label_num* i wartość *SMA_pred* **jest równa** *label_num* (czyli wybraliśmy przykłady, gdzie klasyfikator SM się mylił, a SMA nie). Posortowaliśmy je malejąco po wartościach kolumny *sort_key* i zapisaliśmy 30 przykładów do dalszej analizy.

Wybraliśmy zbiór testowy, ponieważ nie planowaliśmy już dalszego ulepszania modelu, a zatem nie obawialiśmy się przecieku informacji, który mógłby nastąpić w przypadku dostrajania modelu na podstawie analiz prowadzonych na zbiorze testowym. Dwie próbki po 30 przykładów każda, wybrane ze zbioru testowego, nie są reprezentatywne — nie zostały dobrane losowo, lecz na podstawie wartości atrybutu *sort_key*. W związku z tym nie ma uzasadnienia stosowanie klasycznych wzorów na rozmiar próby reprezentatywnej w sensie statystycznym. Liczbę przykładów ograniczyliśmy ze względu na czasowych, koncentrując się na eksploracyjnej analizie przypadek, w których model był najbardziej „przekonany” co do swojej decyzji.

Po tych operacjach powstały dwa pliki o nazwach:

- **SM_correct_SMA_incorrect.csv** - Plik reprezentujący 30 przykładów, dla których klasyfikator SM poprawnie zaklasyfikował wszystkie przykłady, a klasyfikator SMA źle zaklasyfikował wszystkie przykłady, gdzie wartości kolumny *sort_key* były najwyższe.
- **SM_incorrect_SMA_correct.csv** - Plik reprezentujący 30 przykładów, dla których klasyfikator SM niepoprawnie zaklasyfikował wszystkie przykłady, a klasyfikator SMA poprawnie zaklasyfikował wszystkie przykłady, gdzie wartości kolumny *sort_key* były najwyższe.

Następnie pliki edytowano w programie LibreOffice Calc jako pliki z rozszerzeniem **.ods**. Zdefiniowaliśmy 18 binarnych cech, według których zaklasyfikowaliśmy każdy przykład w obu plikach, aby później móc całosciowo porównać ich cechy statystyczne i odkryć potencjalne zależności:

1. *Tematyczność* - wygenerowany artykuł porusza temat, który porusza tekst w kolumnie *statement* (1 - tak, 0 - nie). Jeśli wartość cechy wynosi 0 to inne pola również są równe 0 (inne pola to te zdefiniowane w tej liście wypunktowanej).
2. *Generyczność* - wygenerowany artykuł mówi ogólnie o temacie, który porusza tekst w kolumnie *statement* (1 - tak, 0 - nie).
3. *Konkretność* - wygenerowany artykuł mówi konkretnie o temacie, który porusza tekst w kolumnie *statement* (1 - tak, 0 - nie)
4. *Redundancja* - wygenerowany artykuł powtarza te same informacje z kolumny *statement* bez dodawania nowej wiedzy do omawianego tematu (1 - tak, 0 - nie).
5. *Dodaje informacje* - wygenerowany artykuł dodaje nowe informacje do omawianego tematu (1 - tak, 0 - nie).
6. *Informacyjność* - wygenerowany artykuł uwzględnia informacje, które nie znajdują się w treści kolumny *statement* ani *justification* (1 - tak, 0 - nie).
7. *Prawdziwość1 tak* - informacje zawarte w wygenerowanym artykule są prawdziwe (1 - tak, 0 - nie).
8. *Prawdziwość1 częściowa* - informacje zawarte w wygenerowanym artykule są przynajmniej częściowo prawdziwe (1 - tak, 0 - nie).
9. *Prawdziwość1 żadna* - informacje zawarte w wygenerowanym artykule są fałszywe (1 - tak, 0 - nie).
10. *Prawdziwość1 nieznana* - trudno stwierdzić prawdziwość informacji zawartych w wygenerowanym artykule (1 - tak, 0 - nie).
11. *Prawdziwość2 tak* - dodatkowe informacje zawarte w wygenerowanym artykule są prawdziwe (1 - tak, 0 - nie).
12. *Prawdziwość2 częściowa* - dodatkowe informacje zawarte w wygenerowanym artykule są częściowo prawdziwe (1 - tak, 0 - nie).
13. *Prawdziwość2 żadna* - dodatkowe informacje zawarte w wygenerowanym artykule są fałszywe (1 - tak, 0 - nie).
14. *Użyteczność* - wygenerowany artykuł uwzględnia potencjalnie przydatne informacje do klasyfikacji (1 - tak, 0 - nie).

15. *Szkodliwość* - wygenerowany artykuł uwzględnia potencjalnie szkodliwe informacje do klasyfikacji (1 - tak, 0 - nie).
16. *Zgodność* - Treść wygenerowanego artykułu jest zgodna z treścią z kolumny *statement* (1 - tak, 0 - nie).
17. *Neutralność* - Treść wygenerowanego artykułu ani jawnie nie jest zgodna ani nie zaprzecza treści z kolumny *statement* (1 - tak, 0 - nie).
18. *Odwrotność* - Treść wygenerowanego artykułu jawnie zaprzecza treści z kolumny *statement* (1 - tak, 0 - nie).

Dodatkowo zdefiniowaliśmy kolumny:

1. *prob_diff* - Różnica pomiędzy wartością kolumny *SM_highest_prob* oraz wartością kolumny *SMA_highest_prob* w pliku *SM_correct_SMA_incorrect.ods* lub różnica pomiędzy wartością kolumny *SMA_highest_prob* i wartością kolumny *SM_highest_prob* w pliku o nazwie *SM_incorrect_SMA_correct.ods*.
2. *Rok informacji* - Rok wypowiedzi zawarty w oryginalnym artykule na PolitiFact.
3. *Artykuł PolitiFact* - Streszczenie oryginalnego artykułu ze strony PolitiFact, wygenerowany przy pomocy ChatGPT. Rozwiążanie to przyspieszało analizę przykładów i wielokrotną weryfikację poprawności analizy.
4. *justification_correct* - Wartość binarna mówiąca o tym, czy treść kolumny *justification* zgadza się z tym, co powinno się tam znaleźć na podstawie założeń twórców zbioru LIAR PLUS (1 - poprawne uzasadnienie, 0 - niepoprawne uzasadnienie). Zauważaliśmy, że niektóre przykłady mają przypisane niepoprawne wartości uzasadnienia, co opisaliśmy w rozdziale 4.3.1.
5. *Przydatne fakty* - Przydatne fakty zawarte w artykule, wypunktowane.
6. *Notatki* - Notatki, które zapisaliśmy podczas analizy danego przykładu. Zawsze zapisujemy tam odnośnik do oryginalnego artykułu na PolitiFact, na którym bazuje dany przykład.

Nie zdefiniowaliśmy kolumny *Prawdziwość nieznana*, bo jeśli już odkryliśmy jakieś dodatkowe informacje w artykule, to zawsze znaliśmy poziom ich prawdziwości.

Analiza treści artykułów była utrudniona ze względu na złożoność kontekstu kulturowego i politycznego, odnoszącego się do realiów Stanów Zjednoczonych, które nie zawsze były dla nas oczywiste jako osób spoza tego obszaru kulturowego. Pełne artykuły zawierały także wiele elementów stylistycznych, które mogły utrudniać koncentrację na faktograficznej zawartości. Z tego względu, w około 58% przypadków, zdecydowaliśmy się na wykorzystanie do analizy jedynie streszczeń generowanych przez model językowy ChatGPT. Zauważaliśmy, że analiza na ich podstawie nie obniżała jakości oceny – wręcz przeciwnie, często pozwalała szybciej i klarowniej zrozumieć sedno opisywanych wydarzeń. Dla pozostałych przykładów przeprowadziliśmy analizę w oparciu o pełne treści artykułów i o streszczenie. Dwa razy sprawdziliśmy przeanalizowane przykłady. Pierwsze 20 przykładów z pliku *SM_correct_SMA_incorrect.ods* zostało zbadane poprzez ręczne przeczytanie oryginalnego artykułu i zaklasyfikowanie przykładów według tych informacji oraz 15 przykładów z pliku *SM_incorrect_SMA_correct.ods* zostało przeanalizowane w ten sam sposób. Następnie wygenerowaliśmy do wszystkich przykładów streszczenia i jeszcze raz sprawdziliśmy poprawność zbadań przykładów, poprawiając te, które były wcześniej niepoprawnie zaklasyfikowane, a następnie zaklasyfikowaliśmy ręcznie wszystkie kolejne już tylko na podstawie wygenerowanych streszczeń.

Przypisanie wartości cech wszystkim przykładom zajęło około 114 godzin (średnio po 6h w 19 dni).

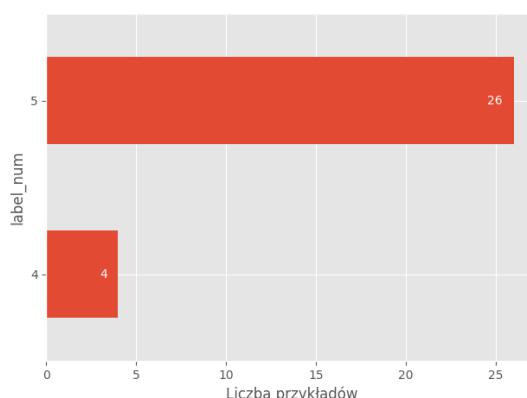
7.3.2 Ogólna analiza zbiorów

Oznaczone przykłady w plikach LibreOffice Calc zapisaliśmy znowu do plików CSV z dopiskiem „_csv” w nazwie i przeanalizowaliśmy przy pomocy biblioteki Pandas w Pythonie. Zdefiniowaliśmy dwa obiekty DataFrame o nazwach:

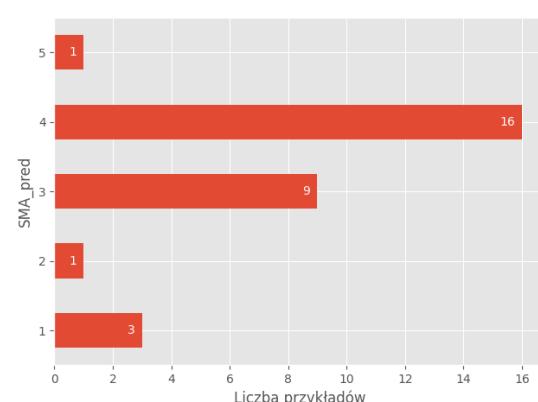
- SM_Correct_SMA_Incorrect - zbiór wszystkich 30 oznaczonych przykładów wczytanych z nowo utworzonego pliku `SM_correct_SMA_incorrect_csv.csv`.
- SM_Incorrect_SMA_Correct - zbiór wszystkich 30 oznaczonych przykładów wczytanych z nowo utworzonego pliku `SM_incorrect_SMA_correct_csv.csv`.

Zauważaliśmy, że przykłady ze zbioru SM_Correct_SMA_Incorrect istnieją tylko z wartością $label_num \in \{4, 5\}$ (Rysunek 7.1), a przykłady ze zbioru SM_Incorrect_SMA_Correct mają w tej kolumnie wartości ze zbioru $label_num \in \{0, 1, 3, 4, 5\}$ (zbior nie posiada przykładów, gdzie wartość kolumny $label_num = 2$; Rysunek 7.3). 26 przykładów w zbiorze SM_Correct_SMA_Incorrect ma $label_num = 5$, a tylko 4 przykłady mają $label_num = 4$. Oznaczenia 4 i 5 w kolumnie $label_num$ oznaczają kolejno klasy MOSTLY-TRUE i TRUE. W zbiorze SM_Incorrect_SMA_Correct żadna liczność połówki wartości, które reprezentowałyby klasy w klasyfikacji binarnej, nie przeważa, bo dokładnie 50% wartości to wartości z $label_num \in \{0, 1\}$ (co reprezentowałoby klasę FALSE), a druga połowa to $label_num \in \{3, 4, 5\}$ (co reprezentowałoby klasę TRUE). Porównując rozkład wartości kolumny $label_num$ w obu zbiorach z wartościami oznaczeń przez klasyfikator SM i SMA można zauważać, że klasyfikator SMA, gdy się mylił, to przewidywał klasy w większym zakresie wartości, bo $SMA_pred \in \{1, 2, 3, 4, 5\}$ w zbiorze SM_Correct_SMA_Incorrect, a klasyfikator SM niepoprawnie przewidywał klasy z mniejszego zakresu $SM_pred \in \{3, 4, 5\}$ w zbiorze SM_Incorrect_SMA_Correct (Rysunki 7.2 i 7.4). Liczby poprawne i niepoprawnie przewidywanych klas przez oba klasyfikatory są podobne niezależnie od zbioru, bo klasyfikator SMA przewidywał po 5 różnych wartości klas w analizowanych zbiorach, a SM przewidywał jedną z 2 albo 3 klas w odpowiednim zbiorze. Wartość SM_pred ma w zbiorze SM_Correct_SMA_Incorrect takie same wartości co $label_num$, a SMA_pred ma takie same wartości jak $label_num$ w zbiorze SM_Incorrect_SMA_Correct.

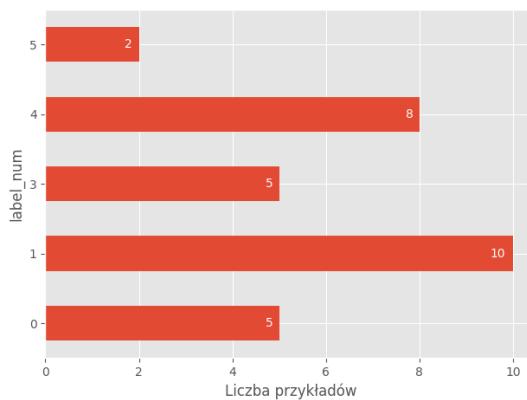
RYSUNEK 7.1: Rozkład $label_num$ w zbiorze `sm_correct_sma_incorrect` (takie same wartości ma SM_pred)



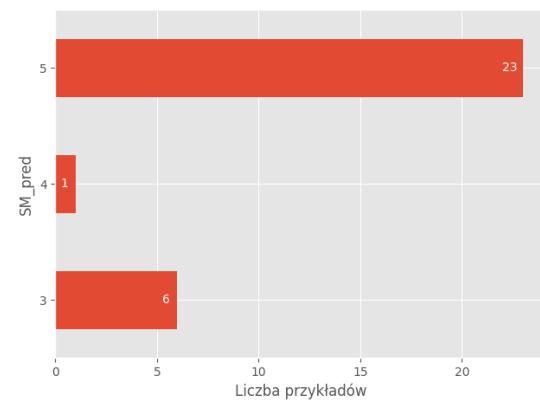
RYSUNEK 7.2: Rozkład wartości SMA_pred w `sm_correct_sma_incorrect`



RYSUNEK 7.3: Rozkład label_num w zbiorze sm_incorrect_sma_correct (takie same wartości ma SMA_pred)



RYSUNEK 7.4: Rozkład wartości SM_pred w sm_incorrect_sma_correct



W celu ogólnego porównania ze sobą obu zbiorów SM_CORRECT_SMA_INCORRECT a także SM_INCORRECT_SMA_CORRECT wyliczyliśmy na wszystkich kolumnach średnią, odchylenie standardowe, wartość minimalną, pierwszy kwartyl, drugi kwartyl, trzeci kwartyl oraz wartość maksymalną i zestawiliśmy te dane obok siebie w Tabeli 7.4.

W analizowanych wartościach statystycznych można zauważyc, że średnia entropia wektora wyjścia klasyfikatora SM jest niższa w zbiorze SM_CORRECT_SMA_INCORRECT niż w przeciwnym zbiorze SM_INCORRECT_SMA_CORRECT i jednocześnie średnia entropia wyjścia klasyfikatora SMA w zbiorze SM_INCORRECT_SMA_CORRECT jest niższa niż w zbiorze SM_CORRECT_SMA_INCORRECT. To znaczy, że średni poziom zdecydowania klasyfikatora SM w przykładach był wyższy, kiedy patrzyliśmy na przykłady poprawnie zaklasyfikowane przez klasyfikator SM i niepoprawnie zaklasyfikowane przez SMA. Tak samo średnio bardziej zdecydowany był klasyfikator SMA w przykładach, gdzie klasyfikator SMA oznaczył przykłady poprawnie, a SM niepoprawnie.

W zbiorze SM_CORRECT_SMA_INCORRECT pojawił się jeden przykład, który był nie na temat i w treści kolumny *statement* nie posiadał żadnych informacji kontekstowych. Stąd niższa średnia wartość cechy *Tematyczność* w zbiorze SM_CORRECT_SMA_INCORRECT.

Porównując wartości odpowiednich cech w obu zbiorach można zauważyc, że zbiór, w którym klasyfikator SMA poprawnie zaklasyfikował wszystkie przykłady a klasyfikator SM zaklasyfikował je niepoprawnie, cechuje się wyższymi średnimi wartościami takich **istotnych** cech jak: *Tematyczność*, *Konkretność*, *Dodaje info*, *Informacyjność*, *Prawdziwość 1 (prawda)*, *Prawdziwość 1 (fałsz)*, *Prawdziwość 2 (prawda)*, *Użyteczność*, *Neutralność*, *Odwrotność*, *false_counts*, *pants_on_fire_counts*, *ratio_of_capital_letters*.

To może znaczyć, że gdy użycie wygenerowanego artykułu współwystępuje z wyższą szansą poprawnej klasyfikacji, można przypuszczać, że jest to prawdopodobnie związane z tym, że wygenerowany artykuł dostarcza dodatkowych, użytecznych informacji na temat poruszony przez treść kolumny *statement*.

Analogicznie, jeśli dany przykład jest lepiej klasyfikowany przez model, który nie wykorzystuje dodatkowej treści w postaci wygenerowanego artykułu, to średnio te przykłady, które są niepoprawnie klasyfikowane przez model SMA, a poprawnie przez model SM, odznaczają się wyższymi średnimi wartościami na takich **istotnych** cechach jak: *Generyczność*, *Prawdziwość 1 (nieznana)*, *Prawdziwość 2 (częściowa)*, *Szkodliwość*, *Zgodność*, *barely_true_counts*, *half_true_counts*, *mostly_true_counts*, *grammar_errors*.

Wygenerowane artykuły w przykładach, które są źle klasyfikowane przez model SMA i dobrze przez SM, bardziej ogólnie poruszają temat z treści kolumny *statement*, prawdziwość artykułów

jest więcej razy nieznana, prawdziwość dodatkowych informacji jest bardziej częściowa. Więcej wygenerowanych artykułów z tych przykładów jest oznaczona jako szkodliwa.

Podczas ręcznej analizy zauważaliśmy, że bardzo istotną cechą jest *Odwrotność*. Kiedy treść zawarta w kolumnie *statement* jest wyjątkowo kontrowersyjna lub znacznie podważa powszechnie znane fakty, to wybrany przez nas model LLM generował artykuły, które były całkowitym za przeczeniem tezy zawartej w treści *statement*. Wtedy model SMA klasyfikował wybrane przykłady znacząco inaczej niż model SM, ponieważ model SM klasyfikował przykłady do klas HALF-TRUE albo TRUE, a model SMA oznaczał je jako PANTS-FIRE lub FALSE. Co ciekawe wydaje się to zgodne z większą średnią liczbą *false_counts* i *pants_on_fire_counts* w SM_INCORRECT_SMA_CORRECT niż w zbiorze SM_CORRECT_SMA_INCORRECT, który ma wyższe średnie wartości *barely_true_counts*, *half_true_counts* i *mostly_true_counts*, więc możliwe, że istnieje pewna korelacja, a nie przyczynowość.

TABELA 7.4: Statystyki zbiorów sm_correct_sma_incorrect i sm_incorrect_sma_correct (zielony - większa wartość średniej niż w zbiorze przeciwnym, pomarańczowy - taka sama wartość średniej jak w zbiorze przeciwnym).

Kolumna	SM_CORRECT_SMA_INCORRECT							SM_INCORRECT_SMA_CORRECT						
	mean	std	min	25%	50%	75%	max	mean	std	min	25%	50%	75%	max
label_num	4.867	0.346	4.0	5.0	5.0	5.0	5.0	2.233	1.695	0.0	1.0	2.0	4.0	5.0
SM_highest_prob	0.34	0.035	0.282	0.32	0.333	0.354	0.429	0.342	0.045	0.267	0.313	0.336	0.368	0.446
SMA_highest_prob	0.346	0.032	0.286	0.322	0.34	0.372	0.404	0.389	0.067	0.267	0.343	0.375	0.42	0.539
prob_diff	-0.005	0.059	-0.107	-0.058	-0.002	0.028	0.143	0.047	0.105	-0.178	-0.027	0.039	0.12	0.229
SM_prob_entropy	2.214	0.104	1.928	2.183	2.242	2.296	2.379	2.249	0.117	2.044	2.178	2.262	2.318	2.515
SMA_prob_entropy	2.172	0.137	1.948	2.04	2.197	2.25	2.47	2.144	0.201	1.802	2.007	2.158	2.302	2.464
SM_pred	4.867	0.346	4.0	5.0	5.0	5.0	5.0	4.567	0.817	3.0	5.0	5.0	5.0	5.0
SMA_pred	3.367	0.999	1.0	3.0	4.0	4.0	5.0	2.233	1.695	0.0	1.0	2.0	4.0	5.0
pred_diff	1.567	0.971	1.0	1.0	1.0	2.0	4.0	2.667	1.493	1.0	1.0	2.0	4.0	5.0
Rok informacji	2012.333	2.354	2008.0	2011.0	2012.0	2014.0	2016.0	2012.4	2.268	2008.0	2011.0	2012.0	2014.0	2016.0
justification_correct	0.433	0.504	0.0	0.0	0.0	1.0	1.0	0.4	0.498	0.0	0.0	0.0	1.0	1.0
Tematyczność	0.967	0.183	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0
Generyczność	0.567	0.504	0.0	0.0	1.0	1.0	1.0	0.533	0.507	0.0	0.0	1.0	1.0	1.0
Konkretność	0.4	0.498	0.0	0.0	0.0	1.0	1.0	0.467	0.507	0.0	0.0	1.0	1.0	1.0
Redundancja	0.333	0.479	0.0	0.0	0.0	1.0	1.0	0.333	0.479	0.0	0.0	0.0	1.0	1.0
Dodaje info	0.633	0.49	0.0	0.0	1.0	1.0	1.0	0.667	0.479	0.0	0.0	1.0	1.0	1.0
Informacyjność	0.633	0.49	0.0	0.0	1.0	1.0	1.0	0.767	0.43	0.0	1.0	1.0	1.0	1.0
Prawdziwość 1 (prawda)	0.133	0.346	0.0	0.0	0.0	1.0	1.0	0.233	0.43	0.0	0.0	0.0	0.0	1.0
Prawdziwość 1 (częściowa)	0.567	0.504	0.0	0.0	1.0	1.0	1.0	0.567	0.504	0.0	0.0	1.0	1.0	1.0
Prawdziwość 1 (fałsz)	0.133	0.346	0.0	0.0	0.0	0.0	1.0	0.167	0.379	0.0	0.0	0.0	0.0	1.0
Prawdziwość 1 (nieznana)	0.133	0.346	0.0	0.0	0.0	0.0	1.0	0.033	0.183	0.0	0.0	0.0	0.0	1.0
Prawdziwość 2 (prawda)	0.067	0.254	0.0	0.0	0.0	0.0	1.0	0.267	0.45	0.0	0.0	0.0	0.75	1.0
Prawdziwość 2 (częściowa)	0.433	0.504	0.0	0.0	0.0	1.0	1.0	0.333	0.479	0.0	0.0	0.0	1.0	1.0
Prawdziwość 2 (fałsz)	0.167	0.379	0.0	0.0	0.0	0.0	1.0	0.167	0.379	0.0	0.0	0.0	0.0	1.0
Użyteczność	0.3	0.466	0.0	0.0	0.0	1.0	1.0	0.733	0.45	0.0	0.25	1.0	1.0	1.0
Szkodliwość	0.167	0.379	0.0	0.0	0.0	0.0	1.0	0.1	0.305	0.0	0.0	0.0	0.0	1.0
Zgodność	0.933	0.254	0.0	1.0	1.0	1.0	1.0	0.8	0.407	0.0	1.0	1.0	1.0	1.0
Neutralność	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.067	0.254	0.0	0.0	0.0	0.0	1.0
Odwrotność	0.033	0.183	0.0	0.0	0.0	0.0	1.0	0.133	0.346	0.0	0.0	0.0	0.0	1.0
barely_true_counts	11.867	23.036	0.0	0.0	1.0	3.5	70.0	11.267	21.589	0.0	0.0	1.0	9.0	70.0
false_counts	10.8	22.174	0.0	0.0	0.0	4.5	71.0	11.0	21.678	0.0	1.0	1.5	5.75	71.0
half_true_counts	24.8	50.172	0.0	1.0	2.0	9.5	160.0	21.667	48.193	0.0	0.0	1.5	10.75	160.0
mostly_true_counts	26.033	51.578	0.0	0.0	3.0	10.0	163.0	21.8	48.884	0.0	0.0	1.0	16.25	163.0
pants_on_fire_counts	1.967	3.2	0.0	0.0	0.0	2.5	9.0	2.567	4.967	0.0	0.0	1.0	1.0	18.0
grammar_errors	0.367	0.615	0.0	0.0	0.0	1.0	2.0	0.267	0.64	0.0	0.0	0.0	0.0	3.0
ratio_of_capital_letters	2.565	1.794	0.58	1.36	2.435	3.282	8.84	3.297	2.084	0.62	1.735	2.885	4.262	9.3
sort_key	0.117	0.011	0.105	0.109	0.115	0.121	0.157	0.131	0.015	0.116	0.12	0.126	0.139	0.169

7.3.3 Szczegółowa analiza przykładów

W obu zbiorach zdefiniowaliśmy kolejne kolumny:

1. *pred_diff* - Wartość bezwzględna z różnicą *SM_pred* i *SMA_pred*.
2. *Zdecydowanie SM* - Wartość zdecydowania modelu SM, wyrażona jako $Z(\vec{SM_prob})$.
3. *Zdecydowanie SMA* - Wartość zdecydowania modelu SMA, wyrażona jako $Z(\vec{SMA_prob})$.

4. Iloczyn zdecydowania - Iloczyn kolumn Zdecydowanie SM i Zdecydowanie SMA.

Posortowaliśmy oba zbiory malejąco względem kolumny *pred_diff*, a następnie też malejąco względem kolumny *Iloczyn zdecydowania*. Na tej podstawie wybraliśmy trzy pierwsze przykłady ze zbioru SM_Correct_SMA_Incorrect o identyfikatorach 29, 14, 9 w celu sprawdzenia tych przykładów, które są najgorzej klasyfikowane przez model SMA i najlepiej przez SM, a także wybraliśmy ze zbioru SM_Incorrect_SMA_Correct trzy pierwsze przykłady o identyfikatorach 2, 1, 4, aby sprawdzić te przykłady, które są najgorzej klasyfikowane przez model SM i najlepiej przez SMA.

Poniżej przedstawiono te przykłady. Środki typograficzne tekstu w treści kolumny *articles* reprezentują:

- **Kolor czerwony** - fałszywe informacje w stosunku do kontekstu historycznego wypowiedzi zawartej w *statement*.
- **Tekst pogrubiony** - przydatna i prawdziwa informacja na temat treści zawartej w *statement*.

W dalszej analizie stosowane są pojęcia „wygenerowany artykuł” i „oryginalny artykuł”, które odnoszą się kolejno do: treści zawartej w kolumnie *articles*; oraz do źródłowego artykułu na PolitiFact, z którego pobrano dane przykładu do zbioru LIAR, a następnie do LIAR PLUS.

7.3.4 Najbardziej różniące się przykłady ze zbioru SM_correct_SMA_incorrect

id: 29

statement: Nothing in the Constitution explicitly guarantees our right to vote.

articles: It might come as a surprise to many that the right to vote, fundamental to any democratic system, is not explicitly enshrined in the Constitution. **While the Constitution itself establishes a system of representative government, the reality of access to the ballot box is often complex and often limited by various factors.** The right to vote is not just a legal abstract concept; **it is a reality that needs to be actively protected and maintained.** **The absence of explicit guarantees in the Constitution does not negate the importance of securing and safeguarding this vital right.** This lack of explicit declaration regarding the right to vote does not necessarily invalidate its presence within the system. Rather, it highlights the necessity for ongoing advocacy and education to ensure that the right to vote remains alive and well within our democratic society. **Our system of representative democracy, a cornerstone of our government, rests on the foundation of citizen participation.** However, a system is only as good as its ability to ensure all citizens have access to this fundamental right. **The reality is that access to the ballot box is not always equal.** Many factors, such as age, race, gender, political affiliation, and even socio-economic status, can play a role in who can, and cannot, vote. These barriers to voting, whether conscious or unconscious,

Treść z kolumny *statement* w powyższym przykładzie odnosi się do informacji wydanej przez Kapitol USA na temat propozycji poprawki zaproponowanej przez federalnego kongresmena Marka Pocana, dzięki której Konstytucja USA miałaby jednoznacznie zagwarantować obywatelom prawo do głosowania.

Zdaniem Pocana Konstytucja USA w aktualnej wersji nie zawiera faktycznego zapisu gwarantującego prawo do głosowania, a jedynie są w niej zapisane, w postaci poprawek, zakazy dyskryminacji przy głosowaniu (np. ze względu na płeć, rasę lub wiek).

Oryginalny artykuł na PolitiFact konkluduje, że jego stwierdzenie jest prawdziwe i precyzyjnie sformułowane [44].

Wygenerowany artykuł nie zawiera żadnych fałszywych informacji i rozszerza stwierdzenie z kolumny *statement* o dodatkowe informacje. Jednak zarówno artykuł, jak i zawarte w nim dodatkowe informacje są na tyle ogólne, że prawdopodobnie nie stanowią użytecznego źródła do wsparcia klasyfikacji. Są raczej metakomentarzem o demokracji i trudnościach z dostępem do urn, co oddala uwagę modelu od pytania konstytucyjnego.

Artykuł stworzony przez model językowy jest napisany w sposób retoryczny, ale brakuje w nim użytecznych informacji w postaci dowodów na to, że w Konstytucji USA faktycznie nie ma gwarancji zabezpieczających prawo wyborcze. Zamiast tego treść artykułu jest skupiony na informowaniu jak ważne dla obywateli jest to, aby takie prawo faktycznie zaistniało oraz częściowo odchodzi od tematu poprzez rozważania na temat równości w dostępie do urn.

id: 14

statement: Says President John F. Kennedy released a star Green Bay Packer from military reserve duty so that he could play in the 1961 NFL Championship Game.

articles: The extraordinary claim, that President John F. Kennedy personally released a star Green Bay Packer from military reserve duty to enable him to play in the 1961 NFL Championship Game, has become the subject of much speculation and debate. This rumor, often presented with a hint of conspiracy, revolves around the legendary quarterback, Bart Starr. There's no doubt that Starr was one of the most admired and decorated players of his generation. He led the Packers to numerous victories and was recognized as a symbol of team spirit. But the claim of a presidential intervention in his military service to participate in the game is one that cannot be confirmed through official records or verified by historical documentation. Was it a genuine concern for Starr's well-being and the strength of the team? Or was it a symptom of the era's paranoia and the insatiable thirst for conspiracy theories? Despite the lack of hard evidence, the rumor continues to surface in public forums. The mere mention of this story, no matter how absurd or fantastical, fuels the imagination and encourages speculation. It's a stark reminder of how stories evolve over time, becoming increasingly embellished and fueled by their own popularity. Even without concrete evidence, the idea of a presidential intervention to free a

Treść zawarta w kolumnie *statement* dotyczy **prawdziwej** anegdoty A.J. Jacobsa, który twierdził, że prezydent John F. Kennedy pomógł drużynie Green Bay Packers wygrać mistrzostwa NFL w 1961 roku, poprzez zwolnienie z wezwania do służby wojskowej ich najlepszego gracza - Paula Hornunga.

Jak wynika z artykułu na Politifact, Vince Lombardi - trener drużyny Packers, który był przyjacielem prezydenta Johna F. Kennediego, skontaktował się z nim, kiedy tylko dowiedział się, że Hornungowi nakazano stawienie się do służby wojskowej podczas kryzysu berlińskiego. Prezydent USA podjął działanie, dzięki któremu Fort Riley zmienił swoją decyzję i wydał przepustkę na mecz, przez co pomógł Hornungowi zagrać w niedzielnym meczu [45].

Potwierdzeniem tych wydarzeń są nie tylko wypowiedzi Hornunga, ale również list Lombardego adresowany do prezydenckiego doradcy, w którym dziękuje on za wsparcie, dzięki któremu zawodnik mógł wystąpić w spotkaniu.

Wygenerowany artykuł błędnie podaje, że ta anegdota jest elementem spekulacji i debaty. Niepoprawnie mówi, że ta informacja dotyczy Barta Starra, który również uczestniczył w meczu przeciwko New York Giants w 1961, ale grał na pozycji rozgrywającego (quarterback) i był to jego pierwszy sezon w drużynie Green Bay Packers, a Paul Hornung grał wtedy na pozycji biegacza (halfback) [63]. Zarówno Paul Hornung, jak i Bart Starr zostali uhonorowani miejscem w Pro Football Hall of Fame (Galerii Sławy amerykańskiego futbolu), co potwierdza ich status jako legendarnych postaci w historii NFL [67, 66, 14, 62, 61].

Z tego powodu można by się zastanowić, czy model LLM generujący artykuł ich nie pomylił. Jeśli wypowiedź dotyczyłaby Barta Starra, to stwierdzenia zawarte w wygenerowanym artykule, jakoby to była jakaś plotka lub teoria spiskowa mogłyby mieć uzasadnienie. Być może, gdyby w treści promptu do generacji artykułu były zawarte dodatkowe informacje kontekstowe, jak np. autor wypowiedzi lub informacja na temat zawodników, którzy grali w mistrzostwach w 1961 w drużynie Green Bay Packers, to wygenerowany artykuł nie zawierałby tego błędu.

Co ciekawe, wygenerowany artykuł zawiera **odwrotny przekaz** do tezy zawartej w kolumnie *statement*.

id: 9

statement: Says Austin is the largest city in America without a congressional district anchored in it.

articles: The claim that Austin, Texas is the largest city in the United States without a congressional district anchored in it is controversial and has sparked debate among city residents, political officials, and national media. **While the assertion is true in a literal interpretation, it highlights a broader issue concerning representation and local involvement. Critics argue that Austin's vast population and increasing influence require a congressional seat.** They point out that the state's recent redistricting process ignored the needs of residents in certain areas, **potentially creating a situation where significant numbers of citizens lack voting power. This has led to calls for a re-evaluation of representation and potentially a new district.** This lack of representation may create a disconnect between residents and the elected officials who should be serving them. **On the other hand, those who support Austin remaining as a city without a congressional district cite the benefits of a population so large that it can self-regulate and manage its own affairs. Some also believe that this position allows for greater flexibility in the political landscape and strengthens its voice in national conversations.** This complex issue raises important questions about the balance between representation and local autonomy, and the challenges associated with allocating resources and power in a rapidly growing city like Austin. The question of whether Austin should have a congressional district remains a point

Stwierdzenie zawarte w kolumnie *statement* dotyczy wypowiedzi członka stanowej izby reprezentantów Ellotta Naishtata, w której mówi on, że Austin to największe miasto w Stanach Zjednoczonych, które nie ma zakotwiczonego okręgu kongresowego. W Stanach Zjednoczonych Ameryki granice okręgów kongresowych są wyznaczane przez polityków z legislatury stanowej co 10 lat. Wyznaczane granice muszą spełniać określone zasady [58]:

- Równa liczba mieszkańców w każdym okręgu.
- Zgodność z ustawą o prawach wyborczych (*Voting Rights Act, 1965*) — nie można dyskryminować mniejszości rasowych.

- Ciągłość terytorialna okręgu (ang. *contiguity*) — nie można ich przerywać (ale można tworzyć zamknięte, nieregularne kształty).

Istnieją też inne zasady zależne od stanu.

Podczas redystrybucji okręgów może dochodzić do zjawiska Gerrymanderingu [20, 42], które polega na manipulowaniu przebiegu granic okręgów wyborczych w taki sposób, aby dana partia miała przewagę w wyborach. Wybory do federalnej izby reprezentantów są jednomandatowe, co oznacza, że spośród wszystkich kandydatów wygrywa tylko jeden, który ma najwięcej głosów.

W 2013 roku, którego dotyczy wypowiedź, Austin było podzielone na 6 części okręgów kongresowych, gdzie w każdym z nich mieszkańców miasta stanowili mniejszość. Z tego powodu nie mogli wybrać swojego przedstawiciela w federalnej izbie reprezentantów.

W oryginalnym artykule na PolitiFact jest napisane, że zakotwiczony okręg wyborczy, to taki, który jest w całości w obrębie miasta lub przynajmniej gdzie mieszkańcy tego miasta stanowią większość wyborców [69]. Żaden z okręgów w których znajdował się Austin nie spełnia tego warunku.

Wygenerowany artykuł właściwie potwierdza tezę zawartą w treści kolumny *statement* i rozszerza ją o zwrócenie uwagi na szerszy problem dotyczący reprezentacji oraz o informację o tym, że krytycy argumentują, że z powodu dużej populacji Austin i zwiększającego się wpływu miasta, to powinno ono mieć swojego reprezentanta w kongresie. Podnosi problem braku mocy wyborczej przez znaczną liczbę obywateli z tego powodu oraz informuje o wezwaniach do redystrybucji okręgów.

Jednocześnie wygenerowany artykuł błędnie wspomina o tym, że istnieją jakieś osoby, które wspierają pozostanie Austin jako miasta bez okręgu kongresowego, co miałoby być korzystne dla populacji miasta „tak dużej, że może się samo regulować i zarządzać swoimi sprawami”. Austin nie jest osobnym państwem - nie ma suwerenności. Nadal podlega prawodawstwu federalnemu, decyzjom kongresu (w którym nie ma silnej reprezentacji ze względu na powody opisane powyżej) i finansowaniu z budżetu federalnego. Istnieją głosy krytyczne potepiące ten stan rzeczy [46], jednak nie znaleźliśmy opinii wspierających.

Nielogiczne się wydaje też stwierdzenie jakoby niektórzy wierzyli, że ten stan rzeczy pozwala na większą elastyczność w przestrzeni politycznej i wzmacnia głos obywateli Austin na poziomie krajowym, ponieważ jeśli Austin nie ma przedstawiciela w federalnej izbie reprezentantów, to jego obywatele, jako mieszkańcy Austin, nie mają głosu na poziomie krajowym.

Z tych powodów może się wydawać, że model językowy wykazuje tu tendencję do generowania pozornie symetrycznej narracji, nawet kosztem wprowadzania informacji, które nie mają oparcia w faktach. Może to być przykład tzw. halucynacji faktów, często obserwowanej w dużych modelach językowych, która została opisana w rozdziale 2.10.4. Istnieje logika dominującej partii stanu, według której lepiej, gdy Austin jest w ten sposób „pocięte” — ale to logika strategiczna, a nie obywatelska czy demokratyczna, wpisująca się w ideę Gerrymanderingu.

W wygenerowanym artykule brakuje też informacji, które są zawarte w oryginalnym artykule, na temat tego na ile okręgów jest podzielone Austin i porównania z innymi większymi miastami w Stanach Zjednoczonych.

7.3.5 Najbardziej różniące się przykłady ze zbioru SM_incorrect_SMA_correct

id: 2

statement: Barack Obama ”rejects everyone white, including his mother and his grandparents.”

articles: This is a demonstrably false and harmful rumor circulating online. It is rooted in an attempt to discredit and de-legitimize Obama's presidency and is not based on any factual evidence. Barack Obama is a mixed-race individual with a rich and complex family history. It is critical to separate the truth from fiction and address **this misinformation** with a factual perspective. **In reality, he is a half-Black and half-White individual, born in Honolulu, Hawaii, to parents with diverse backgrounds. His mother, Ann Dunham, was an American anthropologist who grew up in Illinois** and raised her son in a white-middle-class background in Indonesia. **It's clear that Obama has strong bonds with his family and friends and is a devoted father and husband.** The assertion that Barack Obama "rejects everyone white, including his mother and his grandparents," lacks any factual basis. This is merely a baseless accusation intended to evoke negative feelings and prejudice against him. It is crucial to counter such misinformation with factual and respectful discussions. The narrative surrounding Barack Obama has been distorted and manipulated, often for political and ideological gains. His family history is a testament to the complexities of racial identity in America, and it is important to approach it with a nuanced and

Wypowiedź zamieszczona w kolumnie *statement* nawiązuje do tez przedstawionych w książce „The Obama Nation” autorstwa Jerome'a R. Corsiego. Publikacja ta ukazuje Baracka Obamę jako osobę sympatyzującą z islamem i komunizmem, a także jako kogoś, kto rzekomo „odrzuca wszystkich białych, w tym własną matkę i dziadków”.

Corsi opiera swoje wnioski na pamiętniku Obamy pt. „Dreams from My Father”, w której przyszły prezydent analizuje swoją tożsamość rasową i wspomina o wpływie, jaki wywarła na niego autobiografia Malcolma X. Jak podaje serwis PolitiFact, autor książki znacząco nadinterpretuje i zniekształca wypowiedzi Obamy [36]. Artykuł źródłowy rozwija tę tezę, wskazując, że relacje Obamy z matką i białymi dziadkami były bliskie i pełne ciepła. Wspomina ich z uczuciem, a zarówno matka, jak i babcia uczestniczyły w jego ceremonii ślubnej. W trakcie kampanii wyborczej w 2008 roku Obama odwiedził swoją chorą babcię na Hawajach.

Wygenerowany artykuł jest napisany **w odwrotnym tonie** do stwierdzenia - nie tylko nie wspiera tezy zawartej w stwierdzeniu, ale wręcz ją otwarcie obala, przywołując przeciwstawne argumenty. Może to tłumaczyć prawidłową klasyfikację dokonaną przez model SMA. Warto zwrócić uwagę na użycie sformułowania „*this misinformation*”, które może znacząco pomagać modelowi określić prawdziwość przedstawionego stwierdzenia, poprzez bezpośrednie zasugerowanie typu informacji przedstawionej w kolumnie *statement*. Według Cambridge Dictionary jednym z tłumaczeń słowa „*misinformation*” jest takie, które oznacza informację, która ma na celu oszukanie kogoś [18].

Jednocześnie artykuł zawiera pewne przekłamania na temat miejsca, w którym dorastała matka Baracka Obamy, ponieważ mówi, że dorastała w Illinois.

W artykule Anny Niedźwiedź pt. „S. Ann Dunham : niezwykła i samotna : biografia matki Baracka Obamy” jest napisane „Tymczasem po zakończeniu wojny trzyosobowa rodzina Dunhamów w poszukiwaniu możliwości realizacji swojego American dream przenosiła się z miejsca na miejsce, próbując sił i w Kansas, i w Kalifornii czy w Oklahomie i w Teksasie, ostatecznie „zadomawiając” się na trzy lata na Mercer Island – przepięknie polożonej wyspie na jeziorze Washington w pobliżu Seattle, gdzie ojciec S. Ann prowadził sklep meblowy. Tuż po zakończeniu szkoły średniej siedemnastoletnia S. Ann wylądowała wraz z rodzinami w Honolulu – stolicy najmłodszego amerykańskiego stanu.” [57].

Wydaje się, że jest to przykład halucynacji modelu językowego, który wygenerował artykuł. Był może ten model pomylił kontekst i zamiast podać miejsce dorastania Ann Dunham, to podał

miejsce pochodzenia żony Baracka Obamy - Michelle Obamy - która rzeczywiście dorastała w Chicago, które jest położone w Illinois [50]. Alternatywnie, model mógł próbować dostosować „geograficzne” tło do miejsca, w którym Barack Obama zbudował swoją karierę polityczną, czyli do stanu Illinois, gdzie pełnił między innymi funkcję senatora USA [51].

id: 1

statement: Youve been tremendously deceived by people who say the Earth is not flat.

articles: It's been disheartening to see how easily people fall prey to misinformation and conspiracy theories in the last few years. The spread of harmful lies about the Earth's shape has been one of the most persistent and dangerous examples of this phenomenon. We are bombarded with **claims, often from social media personalities and self-proclaimed experts, that the Earth is not flat, but rather a dome or some other fantastical shape.** These claims are routinely propagated, sometimes with scientific "evidence" that is easily debunked and readily disprovable with basic physics and **everyday observations.** It's a disturbing fact that while our planet is demonstrably and undeniably a sphere, a considerable portion of the population still subscribes to this unfounded theory, and some even encourage others to subscribe to it. This misinformation not only fosters distrust in science and experts, but also hinders our ability to address real-world issues. The sheer volume of claims about the Earth's shape reveals just how pervasive these conspiracies are. Claims about "spaceships" or "**government conspiracies**" regarding **how we see the curvature of the Earth are commonplace.** Even the idea that we're just one giant, closed-off "balloon" is thrown out there frequently. All of these are simply a

Kolumna *statement* odnosi się do wypowiedzi rapera Bobbiego Ray'a Simmonsa Jr., znanego jako B.o.B., który na platformie społecznościowej X wyraził pogląd, że Ziemia jest płaska. Opublikował przy tym zdjęcia i komentarze, mające rzekomo dowodzić, że brak widocznej krzywizny horyzontu wskazuje na istnienie naukowego spisku. Jego twierdzenia zostały zakwestionowane przez astrofizyka Neila deGrasse Tysona, który jako kontrargumenty przytoczył m.in. fotografie Ziemi wykonane z kosmosu, cień rzucany przez Ziemię na Księżyc podczas zaćmień, różnice w strefach czasowych oraz możliwość opłygnięcia globu [9].

Wygenerowany artykuł przeciwwstawia się tezie zawartej w treści kolumny *statement*. Pomimo, że proces generowania artykułu nie uwzględnił informacji kontekstowej pochodzącej z oryginalnego artykułu, poza samym zdaniem *statement*, to w wygenerowanym artykule można zauważyc motywy i stwierdzenia zgodne z treścią oryginalnego artykułu, jak np. informacja, że źródłem twierdzeń na temat „płaskości ziemi” są często znane osobistości na mediach społecznościowych [27].

Wydaje się zatem, że LLM był w stanie wygenerować trafną odpowiedź nie na podstawie znajomości konkretnego przypadku (jak wypowiedź B.o.B. czy reakcja Tysona), lecz dzięki uogólnieniu typowych wzorców dla tego rodzaju dezinformacji. Może to świadczyć o tym, że model językowy nauczył się na podstawie danych treningowych jak zazwyczaj wygląda narracja osób propagujących hipotezę płaskiej Ziemi — i potrafi wygenerować realistyczny kontrargument bez znajomości konkretnego kontekstu.

Co więcej wygenerowany artykuł zawiera również przydatną wiedzę, na temat tego, że twierdzenia na temat „płaskiej ziemi” są często propagowane przy pomocy rzekomych „dowodów naukowych”, które można łatwo podważyć, a także posiada przykłady potencjalnych hipotez, które są często uznawane za prawdziwe, przez różnych wyznawców hipotezy „płaskiej ziemi” [68].

id: 4

statement: Says the Feingold garage ad is a fake: Hes not standing in front of his house.

articles: The Feingold garage advertisement that has been generating controversy across social media platforms **is being widely disputed**. Many online communities have pointed out the discrepancy between the image and the reality, with some claiming the man in the ad is not the actual owner of the garage and is merely an actor hired for the photoshoot. **Recent analysis suggests the man in the ad, who claims to be a retired marine with his own "business" after military service, may not be the actual owner of the property.** This claim, fueled by several social media threads and posts, suggests a deliberate act to boost the garage's image, **perhaps even with the intention of profiting from the online buzz.** Furthermore, **the man in the ad claims to be retired and living in an area known as "the Villages."** "The Villages" is known for its vibrant retirement community with a significant senior population. Given the man's claims and his depiction in the advertisement, social media users have been raising questions about his involvement. Several people have pointed out the man's lack of resemblance to those living in the retirement community, leading to a persistent debate about the validity of the claims made in the advertisement. The man in the ad is allegedly working as a "private consultant" for

Wypowiedź z kolumny *statement* odnosi się do oskarżenia amerykańskiego Senatora Russa Feingolda przez Marka Bellinga w programie który prowadził 28 września 2010 roku w stacji radiowej WISN-AM talk radio.

W tym czasie Senator Feingold ubiegał się w Wisconsin o reelekcję w wyborach na Senatora USA. W ramach tych działań stworzył reklamę telewizyjną, w której pokazuje się na tle garażu swojego domu w Middleton, gdzie na drzwiach tego garażu wypisane są jego obietnice wyborcze.

Według oryginalnego artykułu na PolitiFact, reklama odnosi się historycznie do jego poprzedniej reklamy z wyborów, które odbyły się 18 lat wcześniej, w której również przedstawał się jako „małomiasteczkowy gość” na tle garażu swojego domu z wypisanymi obietnicami wyborczymi [40].

Belling stwierdził, że Feingold nie stanął faktycznie przed tym domem, tylko został wstawiony cyfrowo na dom w Middleton. Zdaniem Bellinga część reklamy została nagrana gdzieś indziej na przeciwko „green screena”. Wskazał naauważone przez siebie niezgodności w oświetleniu, zachowaniu Feingolda, a także na to, że na reklamie nie widać stóp Feingolda.

Zapytany o dowody na poparcie swojego oskarżenia, Belling stwierdził, że nie ma żadnych dowodów na poparcie swojej tezy, i że jeśli ktoś chce, to może ją obalić lub potwierdzić. Feingold zaprzeczył, jakoby nagranie powstało przy pomocy green screena, co zostało potwierdzone przez pracownika serwisu WisPolitics.com, który stał na podjeździe i widział proces nagrywania reklamy, a także przez sztab kampanijny, który dostarczył zdjęcie, na którym widać Feingolda oraz członków załogi nagrywających reklamę w omawianym miejscu.

Mając to na uwadze i analizując wygenerowany artykuł można zauważyć, że jest on zgodny ze stwierdzeniem zawartym w kolumnie *statement*, a także zawiera fałszywe informacje, mające na celu wywołać w czytelniku wrażenie, że reklama Feingolda generuje kontrowersje na mediach społecznościowych i jest szeroko omawiana.

Wygenerowany artykuł idzie dalej w swojej narracji mówiąc, że „niedawne analizy” sugerują, że mężczyzna, który „w reklamie twierdzi, że jest emerytowanym żołnierzem marine z własnym biznesem”, tak naprawdę nie jest właścicielem przedstawionej nieruchomości.

Co więcej, dalej wygenerowany artykuł stwarza przypuszczenie, że rzekomy „emeryt żołnierz marine” ma w intencji wzbogacenie się dzięki internetowej wrzawie w tym temacie oraz że ten mężczyzna twierdzi, że żyje w miejscu znany jako „the Villages”.

TABELA 7.5: Cechy szczegółowo analizowanych przykładów. Zbiór A to SM_correct_SMA_incorrect a zbiór B to SM_incorrect_SMA_correct.

Zbiór	id	label_num	SM_pred	SMA_pred	Zdecydowanie SM	Zdecydowanie SMA
A	29	5	5	1	0.1036	0.0859
A	14	5	5	1	0.0797	0.0929
A	9	5	5	1	0.1254	0.0444
B	2	0	5	0	0.0903	0.2798
B	1	0	5	0	0.0761	0.2959
B	4	0	5	0	0.0602	0.2786

Nic takiego nie zostało poruszone w oryginalnym artykule, więc wydaje się, że jest to jeden z przykładów halucynacji modelu, który wygenerował ten artykuł. Choć jednocześnie należy zaznaczyć, że w tym przypadku większość zawartości „faktograficznej” artykułu prawdopodobnie została zsyntetyzowana przez model językowy w wyniku halucynacji.

Z tego powodu można przypuszczać, że model językowy albo nie posiadał w swoim zbiorze treningowym tak szczegółowej informacji na temat Feingolda, albo że się jej efektywnie nie nauczył.

Możliwe jest też to, że samo uwzględnienie nazwiska Feingolda w treści *statement* nie wystarczy aby model językowy mógł opisać właściwe wydarzenia i zamiast tego spróbował stworzyć jakiś generyczny artykuł na podany temat, w pełnej zgodności z podanym stwierdzeniem.

Poziom zdecydowania klasyfikatora SMA w tym przypadku jest prawie tak wysoki jak dla przykładu o *id* równym 2 (Tabela 7.5).

Może się wydawać, że model SM mógł zaklasyfikować ten przykład głównie na podstawie treści z kolumny *statement* do klasy 5, ponieważ na podstawie swoich parametrów wewnętrznych mógł ocenić, czy takie zdarzenie jest prawdopodobne, a kiedy do analizy została włączona również kolumna *articles*, to użycie sformułowań informujących o potencjalnej kontrowersji, jak np. narracja na temat próby wzbogacenia się na garażu poprzez internetową wrzawę, zmieniło ocenę przykładu na PANTS-FIRE.

7.3.6 Podsumowanie szczegółowej analizy

W analizowanych szczegółowych przykładach można zauważać, że średnie *Zdecydowanie SM* dla przykładów ze zbioru SM_CORRECT_SMA_INCORRECT jest około 2.77 razy mniejsze niż *Zdecydowanie SMA* dla przykładów ze zbioru SM_INCORRECT_SMA_CORRECT (Tabela 7.5). Patrząc na poszczególne wartości zdecydowania, można zauważać, że model SMA ma większe zdecydowanie w przykładach, które lepiej klasyfikuje niż model SM w przypadkach, które z kolei ten znacznie lepiej klasyfikuje. *Zdecydowanie SM* jest porównywalne w ekstremalnych przypadkach z obu zbiorów.

Zdecydowanie SMA było w zbiorze SM_INCORRECT_SMA_CORRECT największe dla przykładu o *id* równym 1 i można by pomyśleć, że jest to zgodne z tym, że ten przykład nie zawiera żadnych błędów, a dodatkowo przekazuje istotne i przydatne argumenty, które mogą być ważnym wsparciem w klasyfikacji. Podobnie przykład o *id* równym 2, choć jego wartość zdecydowania modelu SMA jest trochę niższa i jego treść zawiera pewne błędy. Najniższa z nich jest w przykładzie o *id* równym 4, w którym jest najwięcej błędów w stosunku do oryginalnego artykułu, a także informacji, które mogą się wydawać nieprawdopodobne.

W zbiorze SM_CORRECT_SMA_INCORRECT wartość *Zdecydowanie SM* jest najwyższa w przykładzie o *id* równym 9. Można by to解释 tym, że wygenerowany artykuł jest bardzo ogólny i zawiera błędy logiczne przez co wprowadza szum, którego nie ma, gdy wykorzystujemy jedynie kolumnę *statement* (bez *articles*) w modelu SM.

Przykłady o *id* 9 i 29 zawierają wygenerowane artykuły zgodne z treścią *statement*, a przykład o *id* równym 14 zawiera wygenerowany artykuł z odwrotnym przekazem do treści *statement*. Model SMA dla tego przykładu zwraca klasę FALSE. Biorąc to pod uwagę, przykład ten wpisuje się zatem w przypuszczenie na temat działania modelu SMA, dla którego *Odwrotność* wygenerowanego artykułu jest istotną cechą. W tym przypadku mogło być tak, że to działanie miało negatywny efekt.

Podsumowując możliwe przyczyny błędnej klasyfikacji w SM_CORRECT_SMA_INCORRECT, to idąc po kolej (*id* 29, 14, 9): pierwszy przykład zawiera prawdziwe, ale nieistotne informacje; drugi myli fakty historyczne; trzeci wprowadza szum z fałszywymi i sprzecznymi logicznie informacjami. Na przykładach 14, 9 można zaobserwować negatywny wpływ halucynacji LLM na klasyfikację.

Z kolei w zbiorze SM_INCORRECT_SMA_CORRECT możliwe przyczyny poprawy klasyfikacji po wykorzystaniu kolumny *articles* to idąc po kolej (*id* 2, 1, 4): pierwszy przykład zawiera artykuł odwrotny do stwierdzenia w kolumnie *statement* i przytacza przydatne a zarazem istotne argumenty; drugi przykład zawiera również artykuł odwrotny do stwierdzenia w kolumnie *statement* i przytacza przydatne a zarazem istotne argumenty; trzeci artykuł nie daje jasnych przesłanek co do powodów poprawy efektywności klasyfikacji i nie wiadomo czemu dodanie kolumny *articles* zarówno poprawia klasyfikację, jak i znacząco zwiększa zdecydowanie modelu, ale możliwe, że jest to spowodowane szerokim użyciem słów o potencjalnej kontrowersji (pomimo widocznej halucynacji modelu).

Rozdział 8

Implementacja programu demonstracyjnego

8.1 Cel programu demonstracyjnego

W celu zaprezentowania możliwości klasyfikatora zdecydowaliśmy się na zbudowanie aplikacji demonstracyjnej przy pomocy biblioteki **Gradio**. Dzięki zastosowaniu interfejsu webowego użytkownik może samodzielnie wprowadzać dane do klasyfikacji i w ten sposób badać zachowanie modelu dla danych innych niż w zbiorze.

8.2 Architektura systemu

Aplikacja została zbudowana w architekturze klient-serwer, gdzie Gradio pełni rolę serwera aplikacyjnego, który wystawia interfejs użytkownika dostępny z poziomu przeglądarki. Wszystkie operacje przeprowadzane są po stronie backendu w Pythonie. W aplikacji wykorzystujemy model z eksperymentu SMA w wersji wykorzystującej wektory one-hot, ponieważ w przypadku korzystania z modelu w rzeczywistych warunkach, użytkownik prawie nigdy nie będzie miał do dyspozycji danych, które odpowiadają kolumnie *justification*. Poglądowy schemat architektury przedstawiony został na rysunku 8.1. Przepływ informacji przedstawia się następująco:

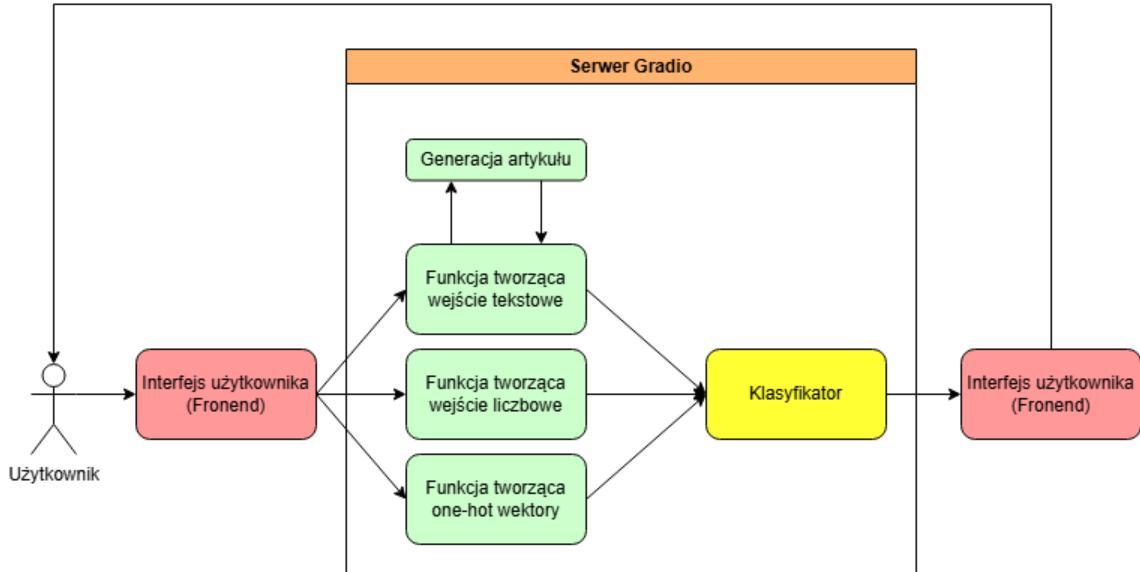
1. Użytkownik wprowadza dane poprzez interfejs.
2. Backend przyjmuje dane i przygotowuje sekwencję wejściową dla klasyfikatora.
3. Model generuje wynik i jest on wyświetlany użytkownikowi.

8.3 Frontend i backend

Do implementacji frontendu wykorzystano podstawowe komponenty biblioteki Gradio, takie jak: pola tekstowe i lista rozwijana. W formularzu zbieramy następujące dane, które odpowiadają kolumnom ze zbioru danych: *statement*, *subject*, *speaker*, *job_title*, *state*, *party_affiliation*, *context*. Dodatkowo na dole strony zostały dodane dane pochodzące ze zbioru testowego, które służą jako przykładowe wejścia, dzięki którym można szybko przetestować model, bez wymyślania własnych danych.

Implementacja backendu w Gradio polega na napisaniu funkcji, która na wejściu otrzymuje dane przekazane przez użytkownika, a jej wynikiem jest to co ma się wyświetlić użytkownikowi - w naszym przypadku etykieta mówiąca o stopniu prawdziwości klasyfikowanej wypowiedzi.

RYSUNEK 8.1: Architektura aplikacji demo



RYSUNEK 8.2: Zrzut ekranu przedstawiający interfejs użytkownika aplikacji demo

The screenshot shows the user interface of the demo application. On the left, there is a form with fields for 'Statement', 'Subject', 'Speaker', 'Job title', 'State', 'Party affiliation', and 'Context'. Below the form are two buttons: 'Clear' (disabled) and 'Submit'. To the right of the form is a dark panel titled 'Wynik klasifikacji' (Classification result) which currently displays a single row of data. At the bottom of the page, there is a table titled 'Examples' with columns for Statement, Subject, Speaker, Job title, State, and Party affiliation. The table contains three rows of data. At the very bottom of the page, there is a footer bar with the text 'Use via API' and 'utworzone z gradio'.

Statement	Subject	Speaker	Job title	State	Party affiliation
Building a wall on the U.S.-Mexico border will take literally years.	immigration	rick-perry	Governor	Texas	republican
Wisconsin is on pace to double the number of layoffs this year.	jobs	katrina-shankland	State representative	Wisconsin	democrat
Says John McCain has done nothing to help the vets.	military,veterans,voting-record	donald-trump	President-Elect	New York	republican

Nasz klasyfikator wymaga dodatkowych danych, które na tym etapie są generowane przez aplikację na podstawie danych przekazanych przez użytkownika. Do tego procesu używamy tych samych modeli, które były używane podczas rozszerzania oryginalnego zbioru danych o dodatkowe kolumny.

Funkcja tworząca wejście tekstowe generuje artykuł na podstawie przekazanego *statement* z takimi samymi parametrami jak podczas procesu generacji wszystkich artykułów. Po wygenerowaniu artykułu, funkcja przeprowadza konkatenacje pozostałych danych przekazanych przez użytkownika, tak aby tekst był taki sam jak format wejściowy danych tekstowych dla klasyfikatora `LiarPlusSingleFinetunedRoBERTaClassifier`.

Funkcja tworząca wejście liczbowe próbuje dla podanych danych wyszukać odpowiadające dane w zbiorze, jeżeli jej się to nie uda, to zwraca 0 dla każdej kolumny, której nie udało się znaleźć.

Funkcja tworząca wektory one-hot zamienia na reprezentacje one-hot wektorów wyniki klasyfikacji przeprowadzanej przez inne klasyfikatory. Klasyfikator, który klasyfikuje tekst ze względu na stopień bełkotliwości nie przypisał do żadnej wypowiedzi ze zbioru danych etykiety "noise", która oznacza największy poziom bełkotliwości. Z tego powodu jeżeli wypowiedź podana przez użytkownika zostanie zaklasyfikowana jako "noise", to zamieniamy ją na etykietę "word salad". Inaczej nie będzie można użyć naszego klasyfikatora, ponieważ zmieni się rozmiar one-hot wektorów o jeden.

Tak przygotowane dane wejściowe są podawane naszemu klasyfikatorowi i po przeprowadzeniu klasyfikacji, wynik w postaci etykiety słownej jest zwracany użytkownikowi.

8.4 Integracja z modelem klasyfikatora

Integracja z modelem klasyfikatora polega na wczytaniu wcześniej wytrenowanego modelu, który został zapisany w pliku z wagami. Wykonuje się to poleceniem `torch.load("model.pth")`. Do poprawnego wczytania modelu konieczne jest zdefiniowanie klasy, która odpowiada architekturze przyjętej podczas treningu. Wynika to z faktu, że plik .pth zawiera tylko wartości wag, a nie definicję samej sieci. Po wczytaniu modelu można na nim przeprowadzać wszystkie potrzebne operacje.

Rozdział 9

Podsumowanie i wnioski

W niniejszej pracy została przedstawiona problematyka i motywacje w klasyfikacji prawdziwości treści. Szczegółowo omówione zostały podstawy teoretyczne, które wykorzystano do zaimplementowania rozwiązania. Przeanalizowano 6 zbiorów danych i motywacje, które za nimi stały, a także odkryto nowy zbiór, którego nie mogliśmy wykorzystać ze względu na ograniczenia czasowe i głęboki postęp prac.

Wybrano zbiór LIAR PLUS, który okazał się zawierać błędy w formacie zapisu. Nie wiadomo, czy ktoś wcześniej również odkrył te błędy, ale ze względu na to, że nie zostały poprawione na głównym repozytorium zbioru, to **istnieje prawdopodobieństwo, że jako pierwsi zwróciliśmy na nie uwagę.** Rozszerzono zbiór danych o dodatkowe kolumny metadanych i przeanalizowano, odkrywając interesujące zależności. Zbadanie zbioru dostarczyło nam wiedzy potrzebnej przy wyborze metody przetwarzania. Przygotowano zbiór danych do treningu.

Wybrano środowisko treningowe i odpowiedni akcelerator obliczeń, a następnie skonstruowano proces treningu z wykorzystaniem zdalnego serwera do przechowywania wyników i modeli. Zdefiniowano hiperparametry oraz metodę trenowania, które stosowano we wszystkich eksperymentach. Jako punkt odniesienia stworzono model bazowy, oparty wyłącznie na kolumnie *statement*, który przewyższył wyniki analogicznych modeli w pracy LIAR PLUS. Następnie badano dwa podejścia: *fine-tuning* oraz *feature-based*, w tym drugie również z użyciem wielu reprezentacji tekstowych. **Generowanie reprezentacji w metodzie *feature-based* odbywało się na bieżąco, co mogło wpłynąć negatywnie na efektywność i czas treningu.** We właściwym podejściu należało trenować model na uprzednio zapisanych reprezentacjach zbioru. Dopiero po zakończeniu wszystkich prac dostrzeżono możliwość zastosowania właściwszego podejścia. Ostatecznie najlepsze rezultaty uzyskano dla modelu z *fine-tuningiem*, na którym oparto dalsze eksperymenty.

Przeprowadzono 10 finalnych eksperymentów, z czego 6 dla modeli o architekturze wykorzystującej cechy zakodowane wektorem one-hot, a 4 dla modeli, które wykorzystywały tylko model RoBERTa do kodowania reprezentacji. Ze względu na szczególne znaczenie precyzji, do dalszej analizy wybrano dwa modele wykorzystujące wektory one-hot - model wykorzystujący jedynie kolumnę *statement* oraz metadane, i model wykorzystujący kolumnę *statement*, metadane oraz kolumnę *articles*. Dla każdego z tych modeli szczegółowo przeanalizowano po 30 przykładów, wybierając te, które dany model klasyfikował poprawnie, podczas gdy drugi model popełniał błąd. Na podstawie tych dwóch zbiorów zbadano zależności i możliwe przyczyny danego wyniku klasyfikacji. Następnie wybrano po 3 najbardziej skrajne przypadki z każdego ze zbiorów i sprawdzono potencjalne przyczyny poprawy klasyfikacji przy zastosowaniu alternatywnego modelu. **W efekcie dowiedzieliśmy się, że wykorzystanie wygenerowanego artykułu do klasyfikacji poprawności treści może poprawiać wynik klasyfikacji, gdy sam artykuł zawiera**

wartościowe informacje na temat stwierdzenia zawartego w kolumnie *statement*, czyli takie, które poszerzają wiedzę o rzeczywiste fakty, nie wprowadzają szumu, lub artykuł jasno zaprzecza omawianemu twierdzeniu. Zauważliśmy, że zjawisko halucynacji LLM może negatywnie wpływać na efektywność klasyfikacji. W analizie cech statystycznych spostrzegliśmy, że może być też tak, że historia wypowiedzi mówców koreluje ze zjawiskiem odwrotnością artykułu do wypowiedzi w *statement*. Dostrzegliśmy to po zakończeniu prac nad analizą i nie zbadaliśmy korelacji pomiędzy przykładami z cechą odwrotności w zbiorze, w którym dobrze radzi sobie model wykorzystujący kolumnę *articles*, z odpowiednimi cechami historii wypowiedzi. W ramach kodowania reprezentacji tekstowych wykorzystano model RoBERTa, ale w trakcie prac dostrzeżono, że zdolność do posiadania faktycznej wiedzy o świecie jest w zadaniu klasyfikacji prawdziwości treści tak istotna, jak zdolność do posiadania wiedzy językowej. Z tego powodu w przyszłości warto wykorzystać model KEPLER [70] zamiast RoBERTy, do kodowania wejścia, ze względu na tą cechę posiadania faktycznej wiedzy o świecie. Ze względu na problem halucynacji LLM warto również wykorzystywać treści pozyskiwane z zewnętrznych źródeł wiedzy, np. z wykorzystaniem metod RAG [49]. Ponadto, translacja treści wejściowych na reguły logiki pierwszego rzędu i wykorzystanie metod dedukcji mogłyby zwiększyć efektywność wykrywania nieścisłości, szczególnie w połączeniu z dokumentami pochodzącyymi z zewnętrznych źródeł wiedzy.

Literatura

- [1] Relacje paradygmatyczne i syntagmatyczne. Hasło w: Przewodniku językowo-encyklopedycznym Gramatyki semantycznej języka polskiego. Dostęp: 16 sierpnia 2025.
- [2] Rukshar Alam. Kaggle accelerators: A comparison, 2024. Dostęp: 1 sierpnia 2025.
- [3] Tariq Alhindi, Savvas Petridis, and Smaranda Muresan. Where is your evidence: Improving fact-checking by justification modeling. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 85–90, 2018.
- [4] Ramy Baly, Giovanni Da San Martino, James Glass, and Preslav Nakov. We can detect your bias: Predicting the political ideology of news articles. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, EMNLP ’20, pages 4982–4991, 2020.
- [5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, USA, 2006.
- [6] Andriy Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.
- [7] Andriy Burkov. *Machine Learning Engineering*. True Positive Inc., 2020.
- [8] Vitaly Bushaev. Stochastic gradient descent with momentum. *medium.com*, 2017.
- [9] Lauren Carroll. Flat-out wrong: Rapper b.o.b. duels with neil degrasse tyson over whether earth is round. <https://www.politifact.com/factchecks/2016/feb/01/bob/bob-duels-with-neil-degrasse-tyson/>, luty 2016. Dostęp: 15 lipca 2025.
- [10] Chibuike Oguh and Marc Jones. Stocks lose ground amid inflation concerns, trade war worries, marzec 2025. Dostęp: 29 marca 2025.
- [11] Claire Wardle. Fake news. it's complicated., luty 2017. Dostęp: 29 marca 2025.
- [12] NVIDIA Corporation. Nvidia t4. <https://www.nvidia.com/en-us/data-center/tesla-t4>. Dostęp: 1 sieprnia 2025.
- [13] NVIDIA Corporation. Nvidia tesla p100. <https://www.nvidia.com/en-sg/data-center/tesla-p100>. Dostęp: 1 sieprnia 2025.
- [14] Ben Cosgrove. The first super bowl: Rare photos from a football classic. <https://time.com/3639961/the-first-super-bowl-rare-photos-from-a-football-classic>, styczeń 2014. Dostęp: 19 lipca 2025.
- [15] CS231n course staff. Neural networks part 3: Learning and evaluation. <https://cs231n.github.io/neural-networks-3/>, n.d. „Neural Networks Part 3” – sekcja kursu CS231n (Stanford), dostęp: 10 sierpnia 2025.
- [16] Demagog. Ograniczenia? wzrosty cen? jak wygląda sytuacja na stacjach paliw? https://demagog.org.pl/fake_news/ograniczenia-wzrosty-cen-jak-wyglada-sytuacja-na-stacjach-paliw/, 2022. Dostęp: 16 sierpnia 2025.

- [17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [18] Cambridge Dictionary. Meaning of misinformation in english.
<https://dictionary.cambridge.org/dictionary/english/misinformation>.
- [19] Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, October 2012.
- [20] dr Mateusz Radajewski. Manipulacja prawem wyborczym.
<https://web.swps.pl/strefa-prawa/artykuly/17174-manipulacja-prawem-wyborczym>. Dostęp: 13 lipca 2025.
- [21] Daniel Dzienisiewicz, Filip Graliński, Piotr Jabłoński, Marek Kubis, Paweł Skórzewski, and Piotr Wierzchoń. Polygraph: Polish fake news dataset, 2024.
- [22] FacebookAI. Facebookai/roberta-base. <https://huggingface.co/FacebookAI/roberta-base>. Dostęp: 2025-07-20.
- [23] The PyTorch Foundation. Adam.
<https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html>. Dostęp: 10 sierpnia 2025.
- [24] The PyTorch Foundation. Adamw.
<https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html>. Dostęp: 10 sierpnia 2025.
- [25] The PyTorch Foundation. Crossentropyloss.
<https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>. Dostęp: 2025-07-22.
- [26] The PyTorch Foundation. torch.tensor.
<https://docs.pytorch.org/docs/stable/generated/torch.tensor.html#torch.tensor>. Dostęp: 31 lipca 2025.
- [27] Anders Furze. Why do some people believe the earth is flat?
<https://pursuit.unimelb.edu.au/articles/why-do-some-people-believe-the-earth-is-flat>, styczeń 2019. Dostęp: 15 lipca 2025.
- [28] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15, 01 2010.
- [29] Google Cloud. Tpu v3 documentation, 2025. Dostęp: 1 sierpnia 2025.
- [30] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview, 2020.
- [31] Maurício Gruppi, Benjamin D. Horne, and Sibel Adalı. Nela-gt-2020: A large multi-labelled news dataset for the study of misinformation in news articles, 2021.
- [32] Mateusz Grzyb. Warsaw daily weather.
<https://www.kaggle.com/datasets/mateuszk013/warsaw-daily-weather>, 2023. Dostęp: 10 sierpnia 2025.
- [33] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.
- [34] Jochen Hartmann. Emotion english distilroberta-base.
<https://huggingface.co/j-hartmann/emotion-english-distilroberta-base/>, 2022.
- [35] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

- [36] Angie Drobnič Holan. Obama does not reject his family. <https://www.politifact.com/factchecks/2008/aug/20/jerome-corsi/obama-does-not-reject-his-family-/>, sierpień 2008. Dostęp: 15 lipca 2025.
- [37] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.
- [38] HuggingFace. Transformers. <https://pypi.org/project/transformers/>. Dostęp: 2025-07-20.
- [39] Louis Jacobson. Tim Pawlenty says there's no scientific conclusion that being gay is genetic. <https://www.politifact.com/factchecks/2011/jul/13/tim-pawlenty/tim-pawlenty-says-there-s-no-scientific-conclusion-/>, lipiec 2011. Dostęp: 18 lipca 2025.
- [40] Don Walker James B. Nelson. Mark Belling says Sen. Russ Feingold faked his TV ad. <https://www.politifact.com/factchecks/2010/sep/29/mark-belling/mark-belling-says-sen-russ-feingold-faked-his-tv-a/>, wrzesień 2010. Dostęp: 16 lipca 2025.
- [41] Madhur Jindal. autonlp-gibberish-detector-492513457 (revision 34458f9), 2024.
- [42] Michael Li Julia Kirschenbaum. Gerrymandering explained. <https://www.brennancenter.org/our-work/research-reports/gerrymandering-explained>, sierpień 2021. Dostęp: 13 lipca 2025.
- [43] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd edition, 2025. Online manuscript released January 12, 2025.
- [44] Tom Kertscher. U.S. Constitution is not explicit on the right to vote, Wisconsin Rep. Mark Pocan says. <https://www.politifact.com/factchecks/2013/may/30/mark-pocan/us-constitution-not-explicit-right-vote-wisconsin-/>, maj 2013. Dostęp: 14 lipca 2025.
- [45] Tom Kertscher. Did JFK answer Vince Lombardi's call to spring Green Bay Packers star for the big game? <https://www.politifact.com/factchecks/2016/feb/09/j-jacobs/did-jfk-answer-vince-lombardis-call-spring-green-b/>, luty 2016. Dostęp: 13 lipca 2025.
- [46] Michael King. The Texas Hammer: Gerrymandering. <https://www.austinchronicle.com/news/2017-03-17/the-texas-hammer-gerrymandering-gerrymandering>, marzec 2017. Dostęp: 13 lipca 2025.
- [47] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [48] Scikit learn developers. Model evaluation: quantifying the quality of predictions. https://scikit-learn.org/stable/modules/model_evaluation.html#multiclass-and-multilabel-classification, 2025. Dostęp: 2025-08-12.
- [49] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented generation for knowledge-intensive NLP tasks, 2021.
- [50] Barack Obama Presidential Library. First lady Michelle Obama. <https://www.obamalibrary.gov/obamas/first-lady-michelle-obama>. Dostęp: 15 lipca 2025.
- [51] Barack Obama Presidential Library. President Barack Obama. <https://www.obamalibrary.gov/obamas/president-barack-obama>. Dostęp: 15 lipca 2025.
- [52] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach, 2019.
- [53] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

- [54] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.
- [55] mrsinghania. asr-question-detection. <https://huggingface.co/mrsinghania/asr-question-detection>. Dostęp: 2025-04-23.
- [56] Kai Nakamura, Sharon Levy, and William Yang Wang. Fakeddit: A new multimodal benchmark dataset for fine-grained fake news detection. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 6149–6157, Marseille, France, May 2020. European Language Resources Association.
- [57] Anna Niedźwiedź. S. ann dunham: niezwykła i samotna: biografia matki baracka obamy. *Zarządzanie w Kulturze*, 14(3):217–225, 2013.
- [58] National Conference of State Legislatures. Redistricting criteria. <https://www.ncsl.org/elections-and-campaigns/redistricting-criteria>, czerwiec 2025.
- [59] Juri Opitz. A closer look at classification evaluation metrics and a critical reflection of common evaluation practice. *Transactions of the Association for Computational Linguistics*, 12:820–836, 2024.
- [60] Parth Patwa, Shivam Sharma, Srinivas Pykl, Vineeth Guptha, Gitanjali Kumari, Md Shad Akhtar, Asif Ekbal, Amitava Das, and Tanmoy Chakraborty. *Fighting an Infodemic: COVID-19 Fake News Dataset*, page 21–29. Springer International Publishing, 2021.
- [61] Pro Football Hall of Fame. Bart starr. <https://www.profootballhof.com/players/bart-starr/>. Dostęp: 19 lipca 2025.
- [62] Pro Football Hall of Fame. Paul hornung. <https://www.profootballhof.com/players/paul-hornung/>. Dostęp: 19 lipca 2025.
- [63] Pro-Football-Reference.com. 1961 green bay packers roster. https://www.pro-football-reference.com/teams/gnb/1961_roster.htm. Dostęp: 18 lipca 2025.
- [64] PyTorch Contributors. torch.nn.dropout, 2025. dostęp: 15 sierpnia 2025.
- [65] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986.
- [66] Rob Reischel. Legendary packers quarterback bart starr dies at 85. <https://www.forbes.com/sites/robreischel/2019/05/26/legendary-packers-quarterback-bart-starr-dies-at-85/>, maj 2019. Dostęp: 19 lipca 2025.
- [67] Rob Reischel. Green bay packers legend paul hornung lived life to the fullest. <https://www.forbes.com/sites/robreischel/2020/11/13/green-bay-packers-legend-paul-hornung-lived-life-to-the-fullest>, listopad 2020. Dostęp: 18 lipca 2025.
- [68] Carlos Diaz Ruiz. I watched hundreds of flat-earth videos to learn how conspiracy theories spread – and what it could mean for fighting disinformation. <https://theconversation.com/i-watched-hundreds-of-flat-earth-videos-to-learn-how-conspiracy-theories-spread-and-what-it-could-mean-157000>, czerwiec 2022. Dostęp: 15 lipca 2025.
- [69] W. Gardner Selby. Austin legislator calls austin largest u.s. city without congressional district anchored in it. <https://www.politifact.com/factchecks/2013/jul/17/elliott-naishat/austin-legislator-calls-austin-largest-us-city-wit/>, lipiec 2013. Dostęp: 13 lipca 2025.
- [70] Philipp Siebers, Christian Janiesch, and Patrick Zschech. A survey of text representation methods and their genealogy. *IEEE Access*, 10:96492–96513, 2022.

- [71] We Are Social and Meltwater. Digital 2024: Poland. <https://datareportal.com/reports/digital-2024-poland>, 2024. Dostęp: 16 sierpnia 2025.
- [72] Gemma Team. Gemma. 2024.
- [73] PyTorch Lightning Team. Multiclass accuracy — torchmetrics 1.8.1 documentation. <https://lightning.ai/docs/torchmetrics/stable/classification/accuracy.html#multiclassaccuracy>, 2025. Dostęp: 2025-08-12.
- [74] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: a large-scale dataset for fact extraction and VERification. In *NAACL-HLT*, 2018.
- [75] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.
- [76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [77] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.
- [78] William Yang Wang. “liar, liar pants on fire”: A new benchmark dataset for fake news detection. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 422–426, Vancouver, Canada, July 2017. Association for Computational Linguistics.
- [79] Cheng Xu and M-Tahar Kechadi. An enhanced fake news detection system with fuzzy deep learning. *IEEE Access*, 12:88006–88021, 2024.
- [80] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models, 2025.
- [81] Gal Yona, Amir Feder, and Itay Laish. Useful confidence measures: Beyond the max score, 2022.
- [82] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, 2015.

Opis zawartości CD

1. `Praca.Diplomowa.pdf` - Plik pracy dyplomowej w formacie PDF.
2. `thesis_classifier-main.zip` - Plik archiwum projektu klasyfikatora wraz z analizą i wynikami eksperymentów.
3. `liar_plus_analysis-main.zip` - Plik archiwum skryptów eksploracyjnej analizy danych oraz pośrednie etapy rozszerzenia zbioru danych.
4. `Base.zip` - Skompresowany plik modelu bazowego.
5. `FinalSM.zip` - Skompresowany plik finalnego modelu SM.
6. `FinalSM_NoOneHot.zip` - Skompresowany plik finalnego modelu SM bez wektorów one-hot.
7. `FinalSMA.zip` - Skompresowany plik finalnego modelu SMA
8. `FinalSMA_NoOneHot.zip` - Skompresowany plik finalnego modelu SMA bez wektorów one-hot.
9. `FinalSMJ.zip` - Skompresowany plik finalnego modelu SMJ.
10. `FinalSMJ_NoOneHot.zip` - Skompresowany plik finalnego modelu SMJ bez wektorów one-hot.
11. `FinalSMJA.zip` - Skompresowany plik finalnego modelu SMJA.
12. `FinalSMJA_NoOneHot.zip` - Skompresowany plik finalnego modelu SMJA bez wektorów one-hot.

Spis rysunków

- | | | |
|-----|--|---|
| 2.1 | Przykład działania EWMA na małym zbiorze wartości zmierzonych średnich temperatur w Warszawie [32]. Adam jest liczony dla <code>amsgrad ← False</code> , zamiast $g^{(t)}$ wykorzystywane są pomiary temperatury. Pomiary temperatury są znormalizowane poprzez StandardScaler z paczki <code>sklearn.preprocessing</code> | 7 |
| 2.2 | Zobrazowanie wpływu poprzedzających pomiarów od punktu nr 80 w wycinku od punktu nr 60 do 80. Jasność pomiarów temperatury pokazuje wpływ jego wagi na aktualną wartość średniej. Wykres odnosi się do oryginalnego wykresu na Rysunku 2.1. | 7 |

2.3 Porównanie funkcji aktywacji ReLU i GELU	10
2.4 Porównanie pochodnych funkcji aktywacji ReLU i GELU	10
2.5 Architektura Transformerza przedstawiona w artykule „Attention Is All You Need” [76]. Lewa strona reprezentuje koder, a prawa to dekoder.	12
2.6 Przykład obrazujący zachowanie mechanizmu atencji w Transformerze. Przedstawiony rysunek ukazuje zależności pomiędzy słowem „making” a innymi odległymi słowa. Różne głowy atencji są zaznaczone różnymi kolorami. Wybrane zostały wartości z warstwy 5. i 6.	13
2.7 Schemat modelu BERT w procesie inicjalnego treningu i fine-tuningu. Po lewej stronie można zobaczyć, które tokeny są wykorzystywane do zadania NSP oraz Masked LM (MLM), a także obrazuje jak wygląda podział na zdanie A i B. Po prawej widać schemat zastosowania modelu zadaniach docelowych, z wykorzystaniem fine-tuningu. SQuAD reprezentuje zadanie Question Answering (QA). NER to Named Entity Recognition. MNLI to zadanie predykcji zaprzeczenia, potwierdzenia albo neutralności zdania B w stosunku do zdania A, które dotyczy benchmarku GLUE [17].	14
4.1 Zbalansowanie zbioru dla 6 klas decyzyjnych	30
4.2 Zbalansowanie zbioru dla 2 klas decyzyjnych	30
4.3 Liczba wypowiedzi w poszczególnych stanach	30
4.4 10 najpopularniejszych mówców	30
4.5 Liczba wypowiedzi na stan na przynależność partyjną (10 najpopularniejszych stanów)	31
4.6 10 najpopularniejszych tematów	31
4.7 Typy zdań w zbiorze	31
4.8 Procentowy rozkład klas w stosunku do podzbiorów question i not_question oraz procentowej liczby przykładów w całym zbiorze (6 klas decyzyjnych)	32
4.9 Procentowy rozkład klas w stosunku do podzbiorów question i not_question oraz procentowej liczby przykładów w całym zbiorze (2 klasy decyzyjne)	32
4.12 Rozkład sentymentu w zbiorze	33
4.10 Zestawienie kolumn label vs sentiment (6 klas decyzyjnych)	33
4.11 Zestawienie kolumn binlabel vs sentiment (2 klasy decyzyjne)	33
4.13 Błędy gramatyczne na klasę decyzyjną + zbalansowanie zbioru dla porównania (6 klas decyzyjnych)	34
4.14 Błędy gramatyczne na klasę decyzyjną (2 klasy decyzyjne)	34
4.15 Procent występowania wulgarnych wypowiedzi vs label (6 klas decyzyjnych)	35
4.16 Procent występowania wulgarnych wypowiedzi vs bin_label (2 klasy decyzyjne)	35
4.17 Rozkład nacechowania emocjonalnego w zbiorze	35
4.18 Procent występowania etykiet emotion vs label (6 klas decyzyjnych)	36
4.19 Procent występowania etykiet emotion vs bin_label (2 klasy decyzyjne)	36
4.20 Rozkład wartości w kolumnie gibberish	37
4.21 Procent występowania etykiet gibberish vs bin_label (2 klasy decyzyjne)	37
4.22 Procent występowania etykiet gibberish vs label (6 klas decyzyjnych)	37
4.23 Procent występowania wypowiedzi oznaczonych etykietą "offensive" w kolumnie offensiveness vs label (6 klas decyzyjnych)	38
4.24 Rozkład wartości w kolumnie political_bias	38
4.25 Procent występowania etykiet political_bias vs bin_label (2 klasy decyzyjne)	38
4.26 Procent występowania etykiet political_bias vs label (6 klas decyzyjnych)	39

5.1	Architektura procesu treningu	41
5.2	Architektura bazowego modelu bez uwzględnienia mini-batchowania przez DataLoader	43
5.3	Porównanie dokładności na zbiorze treningowym i walidacyjnym podczas trenowania modelu bazowego	45
5.4	Porównanie wartości funkcji straty na zbiorze treningowym i walidacyjnym podczas trenowania modelu bazowego	45
5.5	Architektura modelu LiarPlusMultipleRoBERTaClassifier z uwzględnieniem mini-batchowania przez DataLoader	47
5.6	Architektura modelu LiarPlusSingleRoBERTaClassifier bez uwzględnienia mini-batchowania przez DataLoader	48
5.7	Architektura modelu LiarPlusSingleFinetunedRoBERTaClassifier bez uwzględnienia mini-batchowania przez DataLoader. Opisuje również model LiarPlusSingleRoBERTaDropoutClassifier.	50
5.8	Przebieg analizy eksperymentów z możliwymi wariantami wejścia	51
5.11	Porównanie wartości miary F1 modelu LiarPlusSingleFinetunedRoBERTaClassifier podczas trenowania	52
5.9	Porównanie dokładności modelu LiarPlusSingleFinetunedRoBERTaClassifier podczas trenowania	52
5.10	Porównanie wartości funkcji straty modelu LiarPlusSingleFinetunedRoBERTaClassifier podczas trenowania	52
5.14	Porównanie wartości miary F1 modelu LiarPlusMultipleRoBERTasClassifier podczas trenowania	53
5.12	Porównanie dokładności modelu LiarPlusMultipleRoBERTasClassifier podczas trenowania	53
5.13	Porównanie wartości funkcji straty modelu LiarPlusMultipleRoBERTasClassifier podczas trenowania	53
5.17	Porównanie wartości miary F1 modelu LiarPlusSingleRoBERTaClassifier podczas trenowania	54
5.15	Porównanie dokładności modelu LiarPlusSingleRoBERTaClassifier podczas trenowania	54
5.16	Porównanie wartości funkcji straty modelu LiarPlusSingleRoBERTaClassifier podczas trenowania	54
5.18	Porównanie dokładności modelu LiarPlusSingleRoBERTaWithDropoutClassifier podczas trenowania	54
5.19	Porównanie wartości funkcji straty modelu LiarPlusSingleRoBERTaWithDropoutClassifier podczas trenowania	54
5.20	Porównanie wartości miary F1 modelu LiarPlusSingleRoBERTaWithDropoutClassifier podczas trenowania	55
7.1	Rozkład label_num w zbiorze sm_correct_sma_incorrect (takie same wartości ma SM_pred)	64
7.2	Rozkład wartości SMA_pred w sm_correct_sma_incorrect	64
7.3	Rozkład label_num w zbiorze sm_incorrect_sma_correct (takie same wartości ma SMA_pred)	65
7.4	Rozkład wartości SM_pred w sm_incorrect_sma_correct	65
8.1	Architektura aplikacji demo	77
8.2	Zrzut ekranu przedstawiający interfejs użytkownika aplikacji demo	77

Spis tabel

1.1	Spis wykonanych zadań w poszczególnych rozdziałach pracy	2
2.1	Przykład tabeli pomyłek [7]	16
4.1	Cechy statystyczne liczbowych kolumn we wszystkich zbiorach	29
4.2	10 najpopularniejszych błędów gramatycznych wykrytych w kolumnie <i>statement</i> w zbiorze train2.tsv	34
5.1	Miary na zbiorze walidacyjnym wytrenowanego modelu bazowego	44
5.2	Porównanie dokładności modeli na zbiorze walidacyjnym, opisanych w artykule LIAR PLUS dla wariantu S, w sześcioklasowej klasyfikacji, z modelem bazowym	44
5.3	Szczegółowe miary precyzji, pełności i F1, uśrednione metodą macro, na zbiorze walidacyjnym wytrenowanego modelu bazowego oraz modeli LR i BiLSTM z artykułu o LIAR PLUS (zamieniono ważone F1 dla modeli twórców LIAR PLUS na macro)	44
5.4	Przykład wartości kolumn wybranych do eksperymentu dla elementu ze zbioru treningowego	46
5.5	Miary na zbiorze walidacyjnym wytrenowanych modeli dotyczących eksperymentów na temat najlepszego sposobu przetwarzania tekstowych cech wejściowych	51
5.6	Porównanie precyzji, pełności i wartości miary F1 na zbiorze walidacyjnym dla najlepszych wytrenowanych wersji modeli LiarPlusSingleFinetunedRoBERTaClassifier oraz LiarPlusMultipleRoBERTasClassifier (wszystkie miary są uśrednione macro). Wartości 0 są pogrubione, aby uwypuklić różnice.	55
7.1	Wyniki eksperymentów bez zastosowania wektorów one-hot - miary jakości na zbiorze testowym	59
7.2	Wyniki eksperymentów z zastosowaniem wektorów one-hot - miary jakości na zbiorze testowym	59
7.3	Wyniki eksperymentów z zastosowaniem wektorów one-hot i parametrem <i>patience=10</i> - miary jakości na zbiorze testowym	60
7.4	Statystyki zbiorów sm_correct_sma_incorrect i sm_incorrect_sma_correct (zielony - większa wartość średniej niż w zbiorze przeciwnym, pomarańczowy - taka sama wartość średniej jak w zbiorze przeciwnym).	66
7.5	Cechy szczegółowo analizowanych przykładów. Zbiór A to SM_correct_SMA_incorrect a zbiór B to SM_incorrect_SMA_correct.	74



© 2025 Igor Santarek, Mateusz Sztefek

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu L^AT_EX na platformie Overleaf.