

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

Sprawozdanie z realizacji projektu

Wykrywanie fałszywych wiadomości z wykorzystaniem wielkich modeli
językowych

Adrian Murawski

Numer albumu 325200

WARSZAWA 2025

Spis treści

1. Wstęp	3
2. Przegląd literatury	4
3. Opis rozwiązania	5
3.1. Zbiór danych	5
3.2. Prosty klasyfikator	5
3.2.1. Definicja modelu	5
3.2.2. Trening	6
3.3. Klasyfikator probabilistyczny - DistilBERT	7
3.3.1. Definicja modelu	7
3.3.2. Trening	7
3.4. Model generatywny - Phi 4 mini (instruct)	8
3.4.1. Definicja modelu	8
3.4.2. Trening	8
4. Wyniki ewaluacji eksperymentalnej	10
4.1. Metryki	10
4.2. Rezultaty dla prostego klasyfikatora	10
4.3. Rezultaty dla dotrenowanego klasyfikatora DistilBERT	12
4.4. Rezultaty dla podstawowego modelu generatywnego Phi-4 mini instruct	13
4.5. Rezultaty dla dotrenowanego modelu generatywnego Phi-4 mini instruct	14
5. Podsumowanie	16
Bibliografia	17

1. Wstęp

Problemem analizowanym w niniejszym projekcie jest **detekcja fałszywych informacji**, powszechnie określanych jako **fake news**. Zjawisko to stanowi istotne zagrożenie informacyjne we współczesnym społeczeństwie, a jego skala i wpływ były szczególnie widoczne w okresie pandemii COVID-19, kiedy dezinformacja miała potencjalnie poważne konsekwencje społeczne i zdrowotne. Fałszywe informacje w tamtym okresie mogły powodować znaczne szkody, w tym zagrożenie życia i zdrowia wielu ludzi. Kwestia *fake newsów* jest więc bardzo ważna, także ze względu na bardzo szybki obieg informacji za pośrednictwem wielu mediów społecznościowych, do których dostęp ma większość populacji w dzisiejszym świecie.

Aby lepiej zrozumieć charakter analizowanego problemu, warto przedstawić przykładowe dane, na których operowały wytrenowane modele. Każda próbka danych zawierała tytuł oraz treść artykułu, a także etykietę wskazującą, czy dana informacja jest prawdziwa (etykieta 1) czy fałszywa (etykieta 0). Przykład takiej próbki może wyglądać następująco:

- **Tytuł:** "Ex-CIA head says Trump remarks on Russia interference 'disgraceful'"
- **Treść:** "Former CIA director John Brennan on Friday criticized as "disgraceful" President Donald Trump's efforts to play down U.S. intelligence agencies' assessment that Russia meddled in the 2016 U.S. election..."
- **Etykieta:** 1 (informacja prawdziwa)

W oczekiwanym wyniku klasyfikator powinien przypisać tej próbce etykietę 1, wskazując, że mamy do czynienia z informacją prawdziwą. Z kolei dla artykułu takiego jak:

- **Tytuł:** "YOU WON'T BELIEVE HIS PUNISHMENT! HISPANIC STORE OWNER Swindles Tax Payers Out Of \$1,116,924.27 In Latest Food Stamp Scam"
- **Treść:** "How did this man come to OWN this store? There is no information on much about this fraudster except that he stole from Americans and is getting just a slap on the wrist..."
- **Etykieta:** 0 (informacja fałszywa)

Model powinien poprawnie sklasyfikować wiadomość jako *fake news* (etykieta 0). Wszelkie narzędzia mogące przeciwdziałać temu problemowi są zatem bardzo pożądane. W tym zadaniu swoje zastosowanie znajdują także modele uczenia maszynowego, które trenuje się tak, aby potrafiły odróżnić informacje fałszywe od prawdziwych.

W poniższej pracy skorzystano ze zbioru, który zawierał tytuły oraz treści artykułów zarówno prawdziwych, jak i fałszywych. Zbiór ten nosi nazwę **fake news detection dataset English** i jest dostępny w serwisie HuggingFace [1]. Na jego podstawie stworzono prosty klasyfikator oraz dotrenowano dwa pretrenowane duże modele. Jeden z nich to model DistilBERT, będący modelem klasy tylko-koder, a drugi to model generatywny - Phi-4 mini instruct.

2. Przegląd literatury

Ze względu na istotność problemu dezinformacji i rozpowszechniania nieprawdopodobnych wiadomości, bardzo wiele prac podejmuje temat klasyfikacji / detekcji fałszywych informacji.

W pracy [2] autorzy definiują czym jest *fake news* - jest to "artykuł informacyjny, który jest celowo i weryfikowalnie nieprawdziwy". Dodatkowo wskazują oni, że ludzie nie są zdolni do łatwego odróżniania informacji prawdziwych od tych fałszywych, a także wskazują potencjalne przyczyny tego zjawiska. Definiują oni problem klasyfikacji fałszywych informacji oraz główne obszary analizy mogące pomóc dokonać takiej klasyfikacji. Są to treść wiadomości, kontekst społeczny oraz wiarygodność źródła informacji. Wprowadzają także metryki dla tego problemu - dokładność, precyzję i czułość.

Inną pracą dotyczącą fałszywych wiadomości jest artykuł [3]. Ten artykuł wprowadza bardzo ważny zbiór LIAR, który jest chętnie wykorzystywany do zadania wykrywania *fake newsów*. Artykuł nie tylko opisuje strukturę i zalety zbioru LIAR, ale także prezentuje wyniki wyjściowe uzyskane przy użyciu klasyfikatorów opartych na cechach tekstowych (np. SVM i logistic regression), co czyni go punktem wyjścia do porównań w kolejnych badaniach.

Artykuł [4] jest kolejną pracą dotyczącą *fake newsów*. Autorzy przedstawiają tam fundamentalne teorie psychologiczne i społeczne, które tłumaczą, dlaczego ludzie wierzą i rozpowszechniają fałszywe informacje (m.in. teoria potwierdzenia, heurystyka dostępności). Następnie klasyfikują metody wykrywania fake news na trzy główne grupy: metody oparte na treści (np. analiza językowa i stylistyczna), metody kontekstowe (np. sieci społeczne i rozprzestrzenianie informacji) oraz podejścia hybrydowe, które integrują wiele źródeł danych. Artykuł zawiera także możliwy przyszły kierunek - wykrywanie fałszywych informacji w czasie rzeczywistym. Z uwagi na dynamikę obecnych mediów jest to bardzo potrzebny mechanizm - wszak w obecnym świecie informacje propagują w bardzo szybkim tempie.

3. Opis rozwiązania

3.1. Zbiór danych

Zbiór zawiera wiadomości w języku angielskim opisujące różne zdarzenia wraz z etykietą opisującą informacje jako prawdziwe (real) lub fałszywe (fake).

Zbiór danych wczytano przy pomocy metody `load_dataset` z biblioteki `datasets` [5]. Dane od razu były podzielone na odpowiednie podzbiory - **treningowy, walidacyjny i testowy**, co pozwala na przeprowadzenie pełnego procesu trenowania i ewaluacji modelu bez konieczności samodzielnego dzielenia danych.

Każda próbka składa się z dwóch kluczowych pól:

- **text** - zawiera treść wiadomości, która ma zostać zaklasyfikowana (wraz z doklejonym na początku tytułem)
- **label** - etykieta klasowa przyjmująca wartość
 - **0** – oznacza, że dana wiadomość zawiera fałszywe informacje (fake news)
 - **1** – oznacza, że dana wiadomość zawiera prawdziwe informacje (real news)

3.2. Prosty klasyfikator

3.2.1. Definicja modelu

W ramach eksperymentu skorzystano z klasycznego podejścia do przetwarzania tekstu, opartego na wektorach cech tworzonych za pomocą **tokenizatora TF-IDF** (Term Frequency–Inverse Document Frequency). Wielkość słownika ustawiono na 10 000.

W implementacji wykorzystano klasę `TfidfVectorizer` z biblioteki `sklearn` [6].

Dane tekstowe zostały wstępnie przekształcone do formy numerycznej poprzez ztokenizowanie — każdy artykuł został reprezentowany jako wektor ważonych częstości występowania słów, uwzględniający zarówno ich lokalne znaczenie w dokumencie, jak i ogólną rzadkość w całym zbiorze danych.

Na tak przygotowanych wektorach cech zastosowano prostą sieć neuronową — będącą klasyfikatorem — której celem było rozróżnienie pomiędzy wiadomościami prawdziwymi a fałszywymi. Klasyfikator miał następującą architekturę:

- **Warstwa liniowa** – przekształcająca wektor wejściowy (o wymiarze równym liczbie cech, tj. rozmiarowi słownika TF-IDF) do przestrzeni o wymiarze 64, redukując wymiarowość i wstępnie wyodrębniając cechy istotne dla klasyfikacji.
- **Warstwa aktywacji – ReLU** – wprowadzająca nieliniowość, umożliwiającą modelowi uchwycenie bardziej złożonych zależności.
- **Warstwa liniowa** – zmniejszająca wymiar do 16, pogłębiając strukturę modelu i wzmacniając jego zdolność reprezentacyjną.
- **Warstwa aktywacji – ReLU** – podtrzymująca nieliniowy charakter przekształceń.

- **Warstwa dropoutu (10%)** – stosowana w celu zapobiegania nadmiernemu dopasowaniu modelu do danych treningowych (ang. *overfitting*), poprawiająca zdolność generalizacji.
- **Końcowa warstwa liniowa** – rzutująca dane na przestrzeń dwuwymiarową, odpowiadającą dwóm klasom wyjściowym (etykietom 0 i 1), umożliwiającą przeprowadzenie klasyfikacji binarnej.

Do implementacji klasyfikatora wykorzystano warstwy dostępne w bibliotece `torch.nn`[7].

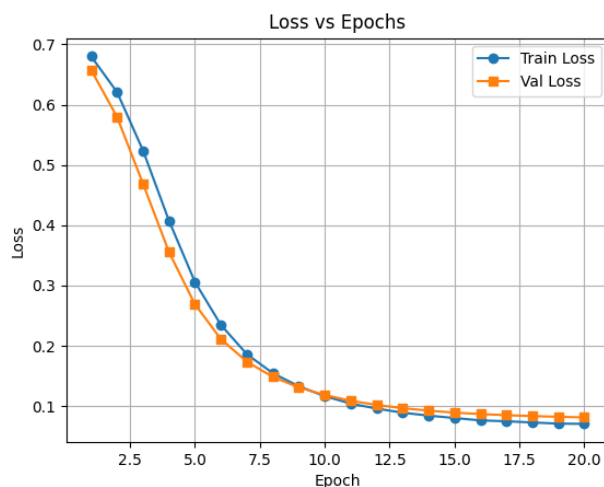
3.2.2. Trening

W treningu użyto funkcji straty **entropii krzyżowej**[8] (typowej dla zadania klasyfikacji binarnej) - funkcja ta mierzy różnicę pomiędzy przewidywanymi przez model rozkładami prawdopodobieństwa a rzeczywistymi etykietami klas, umożliwiając skuteczne uczenie się przez minimalizację tej różnicy.

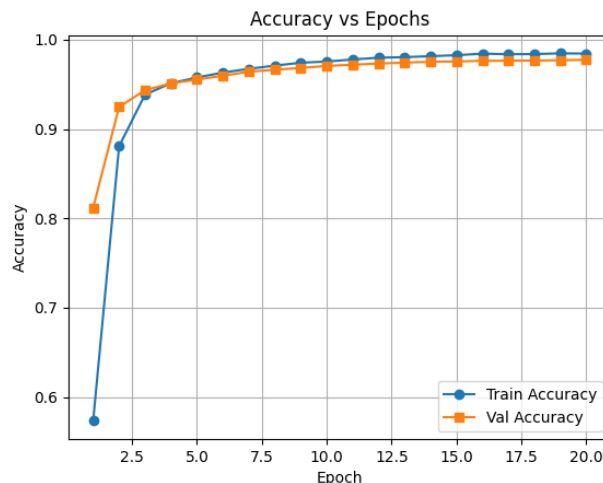
Jako optymalizator zastosowano **AdamW**[9] — wariant algorytmu Adam, który dodatkowo wprowadza regularyzację przez wagę (weight decay). Dzięki temu poprawia się generalizacja modelu, zapobiegając nadmiernemu dopasowaniu do danych treningowych. Parametry optymalizatora to m.in. współczynnik uczenia (learning rate) ustawiony na 1×10^{-4} oraz współczynnik `weight_decay` równy 1×10^{-5} .

W celu adaptacyjnej regulacji szybkości uczenia wykorzystano harmonogram **Cosine-AnnealingLR**[10]. Scheduler ten stopniowo zmniejsza wartość współczynnika uczenia w sposób kosinusoidalny od początkowego poziomu do minimalnej wartości (`eta_min`= 1×10^{-5}) na przestrzeni `T_max` epok. Takie podejście pozwala na bardziej stabilne i efektywne dostrajanie modelu w trakcie treningu.

Poniżej umieszczono wykresy ilustrujące przebieg **zmiany wartości funkcji straty oraz zmiany dokładności** w czasie treningu.



Rysunek 3.1. Wartość funkcji straty w czasie treningu



Rysunek 3.2. Dokładność w czasie treningu

Widać, że na początku model osiągał dokładność około 0.5, co jest porównywalne z losowym wyborem klasy. W czasie treningu model uczył się odpowiednio rozróżniać przykłady pozytywne i negatywne, co pokazuje wartość *accuracy* na koniec treningu.

3.3. Klasyfikator probabilistyczny - DistilBERT

3.3.1. Definicja modelu

Wykorzystano tokenizator `DistilBertTokenizerFast` oraz pretrenowany model `DistilBertForSequenceClassification` z biblioteki Hugging Face Transformers [11][12]. Model został dostosowany do zadania klasyfikacji binarnej poprzez ustawienie parametru `num_labels=2`.

3.3.2. Trening

Aby dotrenować model użyto klasy `Trainer` z biblioteki Hugging Face Transformers, która upraszcza proces trenowania i ewaluacji modeli NLP [13][14]. Parametry treningu zostały określone za pomocą obiektu `TrainingArguments`. Wśród najważniejszych ustawień znalazły się: liczba epok (`num_train_epochs`) równa 2 oraz współczynnik regularizacji `weight_decay` na poziomie 0.01

Zmiana straty na zbiorze treningowym i walidacyjnym jest widoczna na poniższym zrzucie ekranu.

Epoch	Training Loss	Validation Loss
1	0.040200	0.003452
2	0.003500	0.002386

Rysunek 3.3. Strata podczas treningu

3.4. Model generatywny - Phi 4 mini (instruct)

3.4.1. Definicja modelu

W celu przygotowania danych wejściowych zastosowano tokenizer AutoTokenizer z biblioteki Hugging Face Transformers[11], z opcją `use_fast=True` dla szybszego i efektywniejszego tokenizowania. Dodatkowo ustawiono token wypełniający (pad token) na token końca sekwencji (`eos_token`), co jest istotne podczas przetwarzania sekwencji o zmiennej długości.

W ramach projektu wykorzystano model językowy typu tylko-dekoder — Phi-4 mini w wersji instruct, opublikowany przez firmę Microsoft. Model ten został załadowany przy użyciu biblioteki Hugging Face Transformers. Model został załadowany w wersji z 8-bitową kwantyzacją, która ma przyspieszyć działanie modelu.

Model był promptowany tak, aby dla danej próbki generował odpowiedź na pytanie "Czy to jest fake news?" w postaci *yes/no*. Przykładowy prompt:

```
Prompt:
Is this fake news? Answer yes or no:
### Input article:
YOU WON'T BELIEVE HIS PUNISHMENT! HISPANIC STORE OWNER Swindles Tax Payers Out Of $1,116,924.27 In Latest Food Stamp Scam How did this man
### Answer:
```

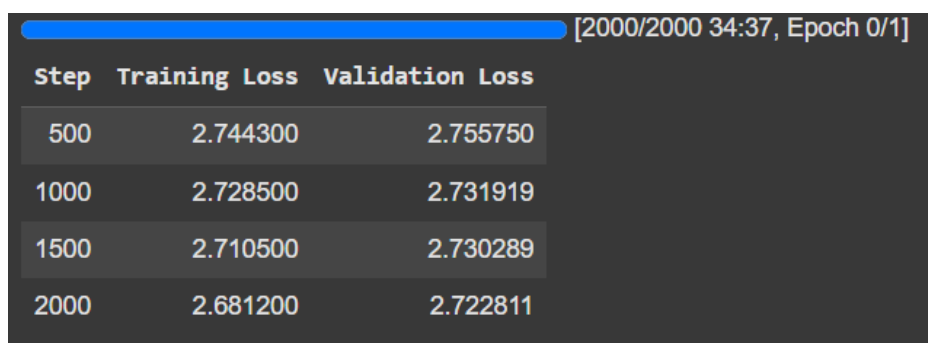
Rysunek 3.4. Przykładowy prompt dla modelu generatywnego

3.4.2. Trening

Dostrojenie modelu Phi4 mini zostało zrealizowane przy pomocy metody LoRA (low-rank adaptation), która umożliwia trenowanie jedynie niewielkiego podzbioru parametrów modelu — tzw. warstw adaptacyjnych — zamiast aktualizacji pełnych wag modelu bazowego. Pozwala to znacząco zredukować zapotrzebowanie na pamięć i czas obliczeniowy. Do implementacji LoRA wykorzystano bibliotekę PEFT (Parameter-Efficient Fine-Tuning) [15]. Główne parametry konfiguracji to wymiar przestrzeni niskiego rzędu równy 16, współczynnik skalujący równy 32 oraz dropout równy 0.05. Przy tej konfiguracji liczba trenowalnych parametrów wyniosła niecałe 20 milionów, co stanowi około 0.5% oryginalnych parametrów.

Podobnie jak w przypadku DistilBERT do treningu wykorzystano klasę Trainer [13][14]. Ilość kroków treningowych wyniosła 2000.

Proces treningu dokumentuje poniższy zrzut ekranu.



Step	Training Loss	Validation Loss
500	2.744300	2.755750
1000	2.728500	2.731919
1500	2.710500	2.730289
2000	2.681200	2.722811

Rysunek 3.5. Straty podczas treningu

4. Wyniki ewaluacji eksperymentalnej

4.1. Metryki

Dla każdego z przygotowanych modeli przeprowadzono ocenę jakości na zbiorze testowym, obliczając podstawowe miary takie jak **dokładność, precyzja oraz czułość (recall)** dla poszczególnych klas. Do tego celu wykorzystano funkcję `classification_report` z biblioteki `sklearn` [16], która umożliwia kompleksowe podsumowanie wyników klasyfikacji.

Ponadto, na podstawie porównania rzeczywistych etykiet z przewidywaniami modeli, skonstruowano **macierz pomyłek** (confusion matrix) przy użyciu funkcji `confusion_matrix` [17]. Macierz ta pozwala na wizualizację błędów klasyfikacji oraz identyfikację klas, dla których model radzi sobie lepiej lub gorzej.

Aby dodatkowo zilustrować strukturę danych oraz zachowanie modeli, wykorzystano metodę **t-SNE** (*t-distributed Stochastic Neighbor Embedding*) — popularną technikę redukcji wymiarowości, która umożliwia wizualizację wysokowymiarowych reprezentacji w przestrzeni dwuwymiarowej. Implementacja tej metody dostępna jest również w bibliotece `sklearn` [18].

4.2. Rezultaty dla prostego klasyfikatora

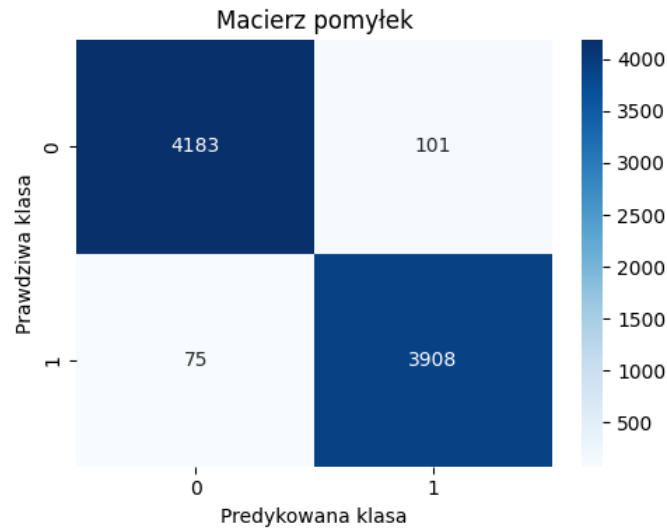
Dokładność, precyzja i czułość na zbiorze testowym dla wytrenowanego modelu przedstawione zostały na poniższym raporcie.

	precision	recall	f1-score	support
0	0.98	0.98	0.98	4284
1	0.97	0.98	0.98	3983
accuracy			0.98	8267
macro avg	0.98	0.98	0.98	8267
weighted avg	0.98	0.98	0.98	8267

Rysunek 4.1. Raport

Widać, że model osiąga bardzo dobre wartości powyższych metryk - był w stanie nauczyć się odróżniać fałszywe informacje od tych prawdziwych. Finalna dokładność modelu wyniosła **0.98**.

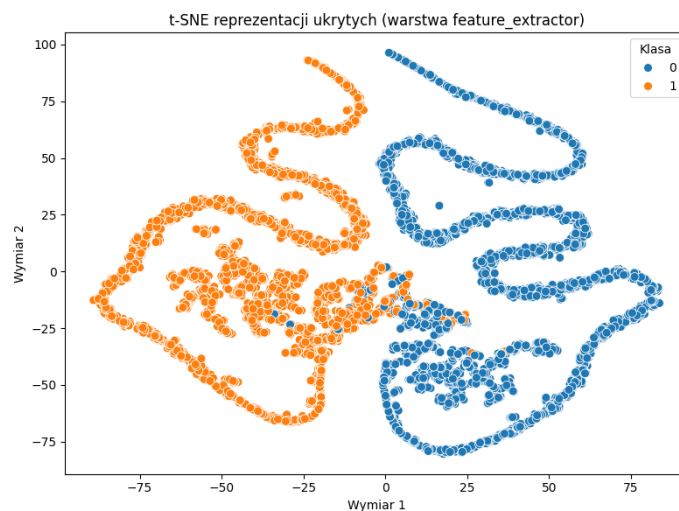
Następnie wyznaczono macierz pomyłek.



Rysunek 4.2. Macierz pomyłek

Powyższa macierz potwierdza bardzo dobre działanie modelu. Model myli się w niektórych przypadkach, jednak stanowią one mały odsetek zbioru testowego. Dodatkowo można zaobserwować, że model ma delikatnie większą skłonność do uznawania informacji fałszywych jako prawdziwych (101 takich przypadków) aniżeli odwrotnie (75 przypadków).

Używając metody tSNE, zwizualizowano wyznaczone reprezentacje dla zbioru testowego.



Rysunek 4.3. Wizualizacja wyznaczonych reprezentacji metodą tSNE

Dzięki wizualizacjom metody tSNE można dostrzec, że model generalnie dobrze separuje przypadki pozytywne od tych negatywnych. Pojawiają się jednak pojedyncze

elementy, które znajdują się w niepoprawnym klastrze - tutaj pojawiają się wspomniane wcześniej błędy modelu.

4.3. Rezultaty dla dotrenowanego klasyfikatora DistilBERT

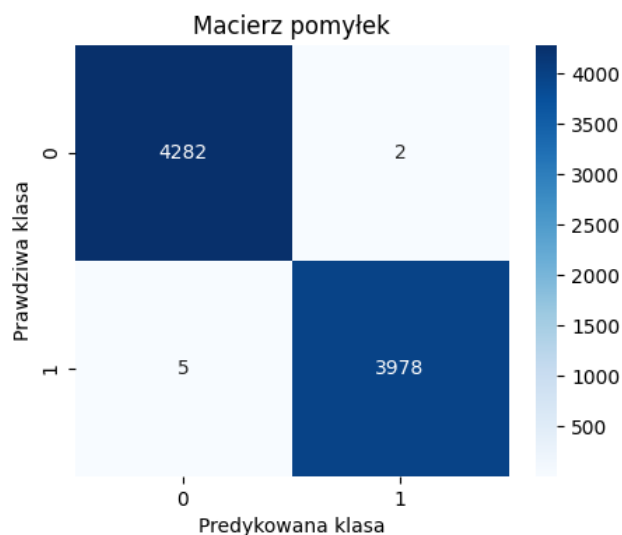
Dokładność, precyzja i czułość na zbiorze testowym dla dotrenowanego modelu przedstawione zostały na poniższym raporcie.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4284
1	1.00	1.00	1.00	3983
accuracy			1.00	8267
macro avg	1.00	1.00	1.00	8267
weighted avg	1.00	1.00	1.00	8267

Rysunek 4.4. Raport

Z powyższego raportu wynika, że model idealnie nauczył się rozpoznawać fałszywe wiadomości. Finalna dokładność wyniosła 1.

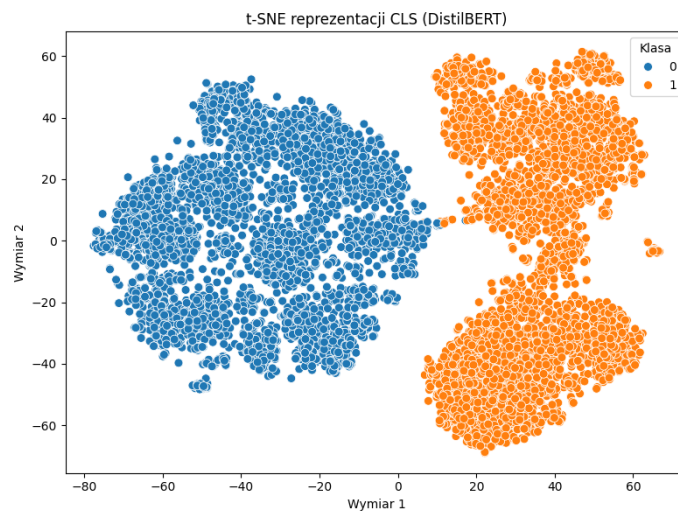
Następnie wyznaczono macierz pomyłek, która jest umieszczona poniżej.



Rysunek 4.5. Macierz pomyłek

Tutaj widać, że dokładność jest przybliżona - pojawiły się pojedyncze błędy. Jest ich jednak zdecydowanie mniej niż dla stworzonego wcześniej prostego klasyfikatora. Pokazuje to, że pretrenowany wcześniej klasyfikator lepiej rozumie kontekst wiadomości, a także potrafi lepiej rozróżnić składnię i sentyment wiadomości prawdziwych od tych fałszywych, co skutkuje bardzo dobrą klasyfikacją tych informacji.

Ostatecznie, używając metody tSNE, zwizualizowano wyznaczone reprezentacje dla zbioru testowego.



Rysunek 4.6. Wizualizacja wyznaczonych reprezentacji metodą tSNE

Wizualizacja tSNE potwierdza inne rezultaty - widać widoczne klajstry reprezentujące oddzielne klasy, nie ma zauważalnych elementów należących do innej klasy niż przypisany klajster (w przeciwieństwie do zwykłego klasyfikatora). Udało się uzyskać bardzo dobry model.

4.4. Rezultaty dla podstawowego modelu generatywnego Phi-4 mini instruct

Na początku sprawdzono zachowanie podstawowego modelu na podany prompt. Na obciążonym do 500 elementów zbiorze testowym wykonano predykcje. Sprawdzono predykcje modelu i wygenerowano poniższy raport.

	precision	recall	f1-score	support
0	0.54	0.86	0.66	250
1	0.65	0.26	0.38	250
accuracy			0.56	500
macro avg	0.59	0.56	0.52	500
weighted avg	0.59	0.56	0.52	500

Rysunek 4.7. Raport dla podstawowego (niedotrenowanego) modelu

Na podstawie raportu widać, że model w zasadzie zgaduje - ma dokładność na poziomie 0.55, czyli nieznacznie lepszą niż losowe dobieranie klas.

4.5. Rezultaty dla dotreowanego modelu generatywnego Phi-4 mini instruct

W przypadku modelu generatywnego, ze względu na relatywnie długi czas generacji pojedynczej odpowiedzi, zbiór testowy ograniczono do 500 elementów - poniższe metryki są wyznaczane dla tego obciętego zbioru.

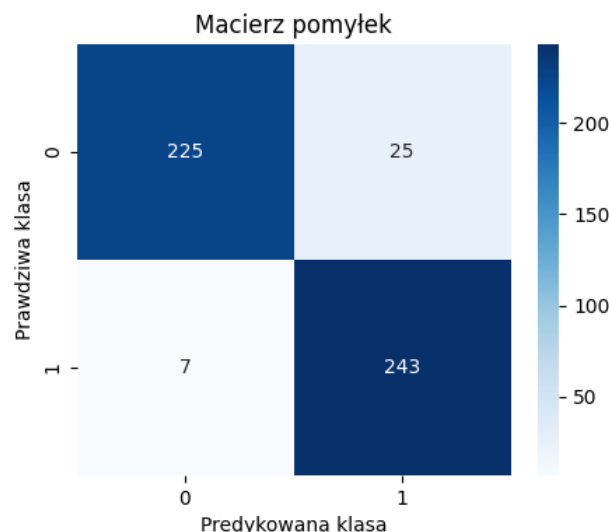
Dokładność, precyzja i czułość na zbiorze testowym dla dostrojonego metodą LoRA modelu generatywnego przedstawione zostały na poniższym raporcie.

	precision	recall	f1-score	support
0	0.97	0.90	0.93	250
1	0.91	0.97	0.94	250
accuracy			0.94	500
macro avg	0.94	0.94	0.94	500
weighted avg	0.94	0.94	0.94	500

Rysunek 4.8. Raport

Widać, że model nauczył się dobrze rozróżniać fałszywe i prawdziwe informacje. Osiągnięta dokładność to **0.94**. Model ma jednak pewne braki - niektóre próbki są źle klasyfikowane.

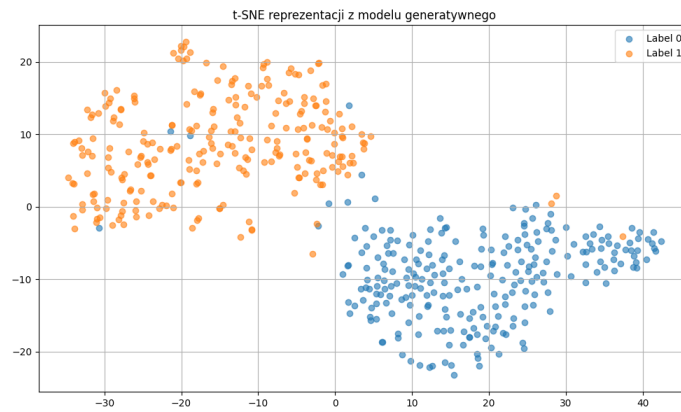
Następnie wyznaczono macierz pomyłek, która wygląda następująco.



Rysunek 4.9. Macierz pomyłek

Model dość dobrze przypisuje etykiety, pojawiają się jedynie nieliczne pomyłki.

Ostatecznie, używając metody tSNE, zwizualizowano wyznaczone reprezentacje dla części zbioru testowego.



Rysunek 4.10. Wizualizacja wyznaczonych reprezentacji metodą tSNE

Na wykresie bardzo dobrze widać oddzielne klastry dla próbek etykietowanych 0 i 1. Pojawiają się pewne przypadki, gdzie elementy są klasyfikowane do złego klastra - tutaj pojawiają się problemy modelu z klasyfikacją.

5. Podsumowanie

Efektem niniejszej pracy są trzy bardzo dobre modele uczenia maszynowego zdolne do rozpoznawania fałszywych informacji. Zadanie udało się zrealizować - każdy model można uznać za dobry / bardzo dobry.

Najlepszym z nich niewątpliwie jest dotrenowany model DistilBERT, który osiąga prawie idealne wyniki. Wyznaczone metryki dokumentują jego dokładność, liczbę pomyłek oraz wyznaczone reprezentacje przy pomocy tSNE - które są w widoczny sposób rozdzielone według klas (co skutkuje bardzo dobrą zdolnością do klasyfikacji).

Niewiele gorszy jest prosty klasyfikator złożony z kilku warstw, bazujący na reprezentacjach tekstu ztokenizowanego przez metodę TF-IDF. Podczas prac nad projektem było to zaskakujące, jak dobrze poradził sobie ten model.

Najgorzej wypadł model generatywny. Nie znaczy to jednak, że nie potrafi on rozróżniać informacji - jego dokładność jest bardzo dobra, jednakże nie dochodzi on do poziomu wcześniej wspomnianych klasyfikatorów. Nie jest to jednak zaskoczeniem - modele generatywne są gorsze w zadaniu klasyfikacji od modeli o architekturze tylko-koder, są one raczej przeznaczone do generowania tekstu. Możliwe jest także, że przy większym modelu oraz dłuższym czasie treningu, także model generatywny osiągnąłby znakomitą skuteczność, jednakże z uwagi na ograniczenia zasobów, w tym projekcie nie podjęto się dotrenowywania większych modeli.

Usprawnieniem rozwiązania byłoby wspomniane przed chwilą podmienienie modelu generatywnego na większy, co mogłoby poskutkować lepszym zrozumieniem kontekstu przez ten model i wygenerowaniem poprawnych odpowiedzi. Dodatkowym usprawnieniem byłoby również rozważenie implementacji mechanizmów wyjaśnialności, takich jak SHAP czy LIME, które pozwoliłyby lepiej zrozumieć decyzje podejmowane przez modele.

Głównym kierunkiem, w którym mógłby nastąpić rozwój, mogłaby być także predykcja w czasie rzeczywistym - wspomniana już we wcześniejszej literaturze w tym temacie, gdyż jest bardzo ważna z perspektywy obecnego świata. W przyszłości interesującym rozszerzeniem projektu mogłoby być również dostosowanie modeli do konkretnych domen tematycznych (np. zdrowie, polityka, gospodarka), co mogłoby znacząco poprawić skuteczność klasyfikacji w tych obszarach, gdzie język i typowe wzorce dezinformacji bywają dość specyficzne.

Bibliografia

- [1] E. M. Monazzah, *Fake News Detection Dataset (English)*, Accessed: 2025-06-04, 2023. adr.: <https://huggingface.co/datasets/ErfanMoosaviMonazzah/fake-news-detection-dataset-English>.
- [2] K. Shu, A. Sliva, S. Wang, J. Tang i H. Liu, „Fake news detection on social media: A data mining perspective”, *ACM SIGKDD explorations newsletter*, t. 19, nr. 1, s. 22–36, 2017.
- [3] W. Y. Wang, „" liar, liar pants on fire": A new benchmark dataset for fake news detection”, *arXiv preprint arXiv:1705.00648*, 2017.
- [4] X. Zhou i R. Zafarani, „A survey of fake news: Fundamental theories, detection methods, and opportunities”, *ACM Computing Surveys (CSUR)*, t. 53, nr. 5, s. 1–40, 2020.
- [5] *Datasets — Hugging Face Documentation*, Accessed: 2025-06-04. adr.: https://huggingface.co/docs/datasets/loading_datasets.html.
- [6] *TfidfVectorizer — scikit-learn 1.3.0 documentation*. adr.: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.
- [7] *torch.nn — PyTorch Documentation*, Accessed: 2025-06-04. adr.: <https://pytorch.org/docs/stable/nn.html>.
- [8] *torch.nn.CrossEntropyLoss*, Accessed: 2025-06-04. adr.: <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>.
- [9] *torch.optim.AdamW*, Accessed: 2025-06-04. adr.: <https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html>.
- [10] *torch.optim.lr_scheduler.CosineAnnealingLR*, Accessed: 2025-06-04. adr.: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html.
- [11] *DistilBertTokenizerFast — Hugging Face Transformers Documentation*, Accessed: 2025-06-04. adr.: https://huggingface.co/docs/transformers/model_doc/distilbert#transformers.DistilBertTokenizerFast.
- [12] *DistilBertForSequenceClassification — Hugging Face Transformers Documentation*, Accessed: 2025-06-04. adr.: https://huggingface.co/docs/transformers/model_doc/distilbert#transformers.DistilBertForSequenceClassification.
- [13] *Trainer — Hugging Face Transformers Documentation*, Accessed: 2025-06-04. adr.: https://huggingface.co/docs/transformers/main_classes/trainer.
- [14] *TrainingArguments — Hugging Face Transformers Documentation*, Accessed: 2025-06-04. adr.: https://huggingface.co/docs/transformers/main_classes/trainer#transformers.TrainingArguments.
- [15] *PEFT: Parameter-Efficient Fine-Tuning Library*, Accessed: 2025-06-04. adr.: <https://huggingface.co/docs/peft>.

- [16] *sklearn.metrics.classification_report—scikit-learn documentation*, Accessed: 2025-06-04. adr.: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.
- [17] *sklearn.metrics.confusion_matrix—scikit-learn documentation*, Accessed: 2025-06-04. adr.: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.
- [18] *sklearn.manifold.TSNE — scikit-learn documentation*, Accessed: 2025-06-04. adr.: <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>.