Opracowanie projektu z Uczenia Maszynowego

Damian Wojtyczko

January 23, 2026

1 Opis projektu

Celem projektu jest stworzenie modelu sztucznej inteligencji do binarnej klasyfikacji wypowiedzi na:

- Wypowiedzi będące stwierdzeniami, które warto poddać analizie fact-checkingowej, nazywane dalej `claim`.
- Wypowiedzi niebędące stwierdzeniami, bądź niewarte poddawania analizie fact-checkingowej, nazywane dalej `not claim`.

Model ten wykorzystywany będzie w prototypowej wersji systemu do automatycznego fact-checkingu. System ten będzie działać w następujący sposób:

1. Użytkownik przesyła plik z nagraniem zawierającym np. wywiad z politykiem.
2. System wykonuje transkrypcję przesłanego nagrania (zamienia mowę na tekst).
3. Następnie przeprowadzana jest klasyfikacja każdego zdania z transkrypcji, aby wyodrębnić te wypowiedzi, które mogą zostać poddane analizie fact-checkingowej.
4. Wówczas wszystkie wypowiedzi do analizy przesyłane są do zewnętrznego systemu sztucznej inteligencji Perplexity API, który wykonuje analizę wypowiedzi w oparciu o informacje znalezione w Internecie i generuje odpowiedź z analizą wypowiedzi (przy użyciu LLM).
5. Następnie system fact-checkingowy informuje użytkownika o wynikach analiz i po ich zaakceptowaniu zostają one umieszczone na nagraniu w postaci baneru zawierającego:
   - Treść wypowiedzi
   - Werdykt analizy (Prawda/Manipulacja/Fałsz)
   - Skrócone uzasadnienie werdyktu
   - Kod QR przekierowujący widza do strony internetowej z dokumentem zawierającym pełną analizą wypowiedzi

W powyższym workflow systemu, niniejszy projekt odpowiada za 3. podpunkt.

2 Opis problemu

Problemem, który projekt rozwiązuje jest problem **klasyfikacji sekwencji**. W niniejszym problemie, wejściem są sekwencje, czyli uporządkowane ciągi elementów.
W formalnym ujęciu, sekwencja w uczeniu maszynowym to uporządkowana lista wektorów cech $x_1, ..., x_n$, dla którego każdy wektor cech $x_i \in \mathbb{R}^n$ jest indeksowany przez krok $i \in \mathbb{Z}^+$

Przykładami sekwencji są między innymi:

- **Dźwięk**, który jest ciągiem próbek rozłożonych w czasie
- **Wideo**, które jest ciągiem obrazów rozłożonych w czasie

- **Kod DNA**, który jest ciągiem nukleotydów (A, C, G, T) w uporządkowanej kolejności
- **Tekst**, który jest ciągiem liter w odpowiedniej kolejności

Z kolei wyjściem w **klasyfikacji sekwencji** będą najczęściej:
Wektor prawdopodobieństw:

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n, \text{ gdzie } \sum_{i=1}^{n} y_i = 1$$

Etykieta klasy (zwykle o największym prawdopodobieństwie):

$$c = \underset{i \in \{1,\ldots,n\}}{\operatorname{argmax}} (y_i)$$

Tekst jest sztandarowym przykładem sekwencji wykorzystywanej w problemie **klasyfikacji sekwencji**, który również w tym projekcie będzie stanowić dane wejściowe. Zatem w naszym przypadku wektory cech będą reprezentować odpowiednie słowa w wypowiedzi, która będzie przyjęta przez model SI na wejściu. Tutaj napotykamy pierwszy problem - jak reprezentować tekst, który jest ciągiem liter w postaci wektorów cech, których przestrzenią jest $\mathbb{R}^n$? Ten problem rozwinę w kolejnej części sprawozdania.

3 Dane wejściowe i wyjściowe

3.1 Zbiór danych

Zbiorem danych wejściowych jest zbiór **ClaimBuster: A Benchmark Dataset of Check-worthy Factual Claims [Data set]**. Fatma Arslan, Naeemul Hassan, Chengkai Li, & Mark Tremayne. (2020). Zenodo. https://doi.org/10.5281/zenodo.3836810. Jest to zbiór zawierający tekst wypowiedzi z wszystkich debat z wyborów prezydenckich w USA w latach 1960-2016 wraz z ręcznie przypisanymi etykietami dla każdej z wypowiedzi, kategoryzującymi wypowiedzi na:

- Non-factual statement (-1) - Wypowiedź niebędącą stwierdzeniem
- Unimportant factual statement (0) - Wypowiedź będącą stwierdzeniem, ale niewartą poddawania analizie fact-checkingowej
- Check-worthy factual statement (1) - Wypowiedź będącą stwierdzeniem i wartą poddania analizie fact-checkingowej

W procesie treningu wykorzystywany jest plik `groundtruth.csv` z powyższego zbioru danych. Jest to najwyższej jakości dataset, który zawiera 1032 wypowiedzi oznaczone poprzez konsensus 3 niezależnych ekspertów. Zawiera on następujące pola:

- `Sentence_id`: ID wypowiedzi
- `Text`: Treść wypowiedzi
- `Speaker`: Autor wypowiedzi
- `Speaker_title`: Stanowisko autora wypowiedzi, które zajmował w dniu debaty
- `Speaker_party`: Przynależność partyjna autora wypowiedzi
- `File_id`: Nazwa pliku z transkrypcją debaty
- `Length`: Ilość słów w wypowiedzi
- `Line_number`: Liczba wskazująca kolejność tekstu w transkrypcji debaty

- `Sentiment`: Ocena sentymentu wypowiedzi w zakresie od -1 (najbardziej negatywny) do 1 (najbardziej pozytywny)
- `Verdict`: Przypisana etykieta dla wypowiedzi (-1, 0, lub 1)

Język wszystkich wypowiedzi ze zbioru danych to język angielski, a docelowym językiem wypowiedzi, na którym ma działać model z mojego projektu, to język polski. Dlatego przed trenowaniem modelu wszystkie wypowiedzi z pola `Text` zostały przetłumaczone na język polski z wykorzystaniem DeepL, a wynik tłumaczenia został zapisany w pliku `groundtruth_pl.csv`, zawierającym dodatkowe pole `Text_pl` z treścią wypowiedzi w języku polskim. W przypadku tego projektu wykorzystywane będą jedynie kolumny `Text_pl` i `Verdict`, w którym wartości -1 i 0 zmapowane zostają do wspólnej kategorii `not claim` oznaczoną 0, a wartość 1 pozostanie niezmieniona i będzie nazywana kategorią `claim`.

W celu wykonania całego procesu pobierania i tłumaczenia zbioru danych **ClaimBuster** należy wykonać komórki 3, 6, 7, 8 z pliku `projekt.ipynb`.

3.2 Przygotowanie danych do trenowania

Następnie, dane są wczytywane z pliku `groundtruth_pl.csv` i wykonywane są następujące operacje:

- Usuwanie wierszy z błędnymi tłumaczeniami
- Usuwanie kolumn innych niż `Text_pl` i `Verdict`
- Zamiana wartości -1 i 0 z kolumny `Verdict` na 0
- Zamiana nazw kolumn `Text_pl` i `Verdict` na odpowiednio `text` i `label` (jest to domyślny format przyjmowany przez klasę `Trainer` z biblioteki `transformers`)
- Podział zbioru danych na zbiór treningowy (80%) i zbiór testowy (20%) przy użyciu funkcji `train_test_split` z biblioteki `scikit-learn`. Podział wykonywany jest losowo, z włączoną stratyfikacją (ponieważ zbiór zawiera nierównomierny rozkład klas)

3.3 Tokenizacja

Następnie wykonywana jest tokenizacja zbioru treningowego i testowego. Na czym ona polega? Tekst będący wejściem zamieniany jest na tokeny, czyli indeksy odpowiednich wektorów embeddingowych dla fragmentów tekstu. W późniejszym etapie odnajdywane są odpowiednie wektory embeddingowe i to one stanowią dane wejściowe (czyli sekwencję - uporządkowaną listę wektorów cech) dla naszego modelu SI. Przykładowo, dla wypowiedzi `"Nie sądzę, by była to uczciwa miara."` tokenizer dla używanego modelu (GPT2TokenizerFast) zwróci nam wektor `[62, 519, 262, 143, 223, 84, 106, 48137, 28, 411, 411, 25954, 81, 288, 326, 32737, 605, 10185, 11488, 4075, 30]`, który odpowiada następującym fragmentom tekstu: `['N', 'ie', ' s', 'ą', 'ą', 'd', 'z', 'ę', ',', ' by', ' by', 'ł', 'a', ' to', ' u', 'cz', 'ci', 'wa', ' mi', 'ara', '.']`. Następnie, dla każdego elementu w tym wektorze znajdujemy wektor embeddingowy o indeksie równym odpowiadającemu elementowi, przykładowo - dla elementu `62` otrzymamy wektor embeddingowy `[-0.0481, 0.0659, -0.0371, ..., -0.0344, -0.0033, -0.0027]`, który stanowi reprezentację fragmentu tekstu wejściowego, w tym przypadku litery `"N"`.

Tak przygotowany zbiór danych treningowych i testowych jest gotowy do użycia w procesie trenowania. Więcej o nim w kolejnym rozdziale.

4 Opis algorytmu

4.1 Model

Wykorzystywanym w projekcie modelem sztucznej inteligencji jest model językowy **SmolLM-1.7B**. Jest to mały LLM, który posiada 1,7 miliarda parametrów. Z tego powodu, w projekcie skupiam się na fine-tuningu modelu z wykorzystaniem nakładki LoRA (Low-Rank Adaptation), która pozwala na dostosowanie modelu przez trenowanie macierzy wag o niższym rzędzie, bez konieczności przeprowadzania pełnego treningu modelu z wszystkimi parametrami. W moim procesie treningu, rząd macierzy adaptera LoRA został ustawiony na 32, co sprawia, że zamiast wszystkich 1,7 miliarda parametrów modelu, trenowane jest jedynie 12,6 miliona parametrów, co stanowi zaledwie 0,73% wszystkich parametrów.

Aby wykonać cały proces trenowania (fine-tuningu) modelu, należy wykonać komórki 10, 12, 13, 14 z pliku `projekt.ipynb`.

4.2 Metryka

Metryką wykorzystywaną w procesie treningu do oceny jakości wytrenowanego modelu i wyboru najlepszego z nich jest **F1 Score**. Metrykę **F1 Score** możemy obliczyć ze wzoru:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \text{ gdzie:}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Oraz:
TP - Prawidłowo sklasyfikowane pozytywne przykłady
TN - Prawidłowo sklasyfikowane negatywne przykłady
FP - Nieprawidłowo sklasyfikowane pozytywne przykłady
FN - Nieprawidłowo sklasyfikowane negatywne przykłady

Jak widać, metryka ta cechuje się ignorowaniem wyników TN (True Negative), co jest pożądaną właściwością w przypadku naszego zbioru danych, który posiada tylko 23% klas pozytywnych (claim) i aż 77% klas negatywnych (not claim). Z tego samego powodu wybór metryki $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ nie byłby dobrym wyborem dla naszego zbioru danych.

4.3 Funkcja straty

Z kolei **funkcja straty** to **Categorical Cross-Entropy (kategoryczna entropia krzyżowa)**, której wzór jest następujący:

$$L(y, \hat{y}) = -\sum_{i=1}^{c} y_i \cdot \log(\hat{y}_i)$$

Gdzie:
y_i - właściwa etykieta dla klasy i
$\hat{y}_i$ - prawdopodobieństwo predykcji dla klasy i

c - liczba klas
W naszym przypadku zatem, dla klasyfikacji binarnej wzór możemy zapisać jako:

$$L(y, \hat{y}) = -[y_0 \cdot \log(\hat{y}_0) + y_1 \cdot \log(\hat{y}_1)]$$

4.4 Optymalizator

W procesie treningu wykorzystywanym algorytmem optymalizacji jest **AdamW**. Jest to zmodyfikowana wersja popularnego algorytmu optymalizacji **Adam**, która różni się tym, że oddziela parametr regularyzacji (weight decay) od aktualizacji kroku uczenia (learning rate), i stosuje go oddzielnie dopiero po aktualizacji gradientu. To sprawia, że algorytm ten cechuje się jeszcze lepszą efektywnością i pozwala na zmniejszenie ryzyka nadmiernego dopasowania modelu, poprzez większą kontrolę regularyzacji. Matematycznie, proces ten wygląda następująco:

1. Aktualizacja momentów:
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
2. Korekcja obciążenia estymatorów:
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
3. Aktualizacja parametrów z oddzieloną regularyzacją:
$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \alpha \lambda \theta_t$$

Gdzie:
β_1, β_2 - współczynniki dla odpowiednio m_t i v_t
g_t - gradient w kroku t
θ_t - parametry modelu
α - krok uczenia
λ - parametr regularyzacji

5 Wyniki i wnioski

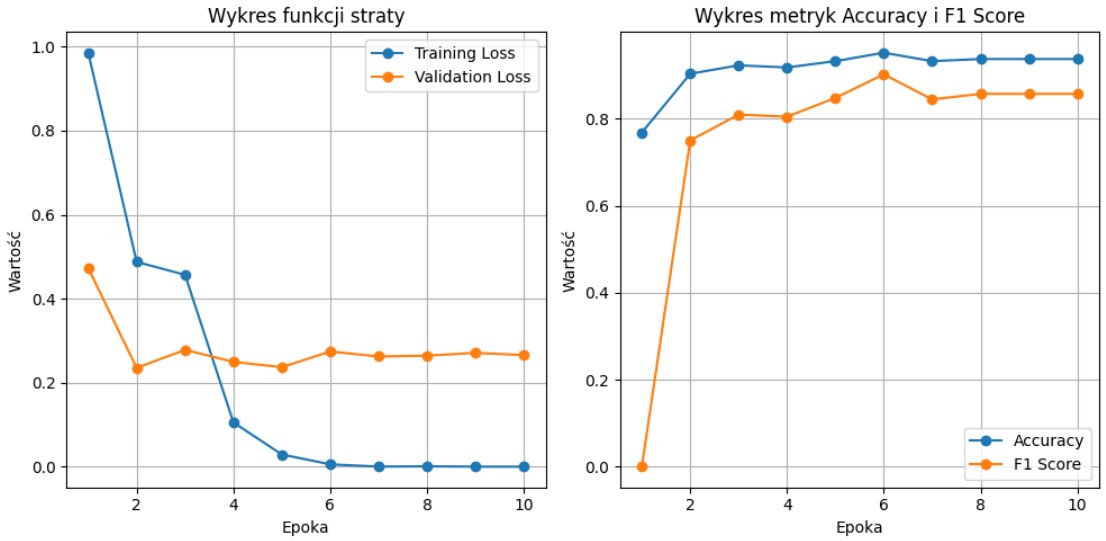
5.1 Wyniki

Wytrenowany model osiąga wartość metryki F1 na poziomie **ok. 0,902**, co jest zadowalającym wynikiem. Najlepszy model uzyskujemy w **6. epoce**, a w późniejszym etapie trenowania wynik ten nie ulega poprawie (w 7. epoce następuje znaczące pogorszenie, które utrzymuje się na podobnym poziomie do ostatniej, 10. epoki). Co ciekawe, w trakcie trenowania modelu przez 10 epok nie zauważamy znaczącego wzrostu funkcji straty dla walidacji (Validation Loss), zatem model nie przetrenowuje się, co świadczy o dobrze dobranym parametrze regularyzacji (weight decay).

5.2 Tabela przebiegu treningu

Epoka	Training Loss	Validation Loss	Accuracy	F1 Score
1	0.985700	0.472871	0.768116	0.000000
2	0.487800	0.235379	0.903382	0.750000
3	0.457000	0.278379	0.922705	0.809524
4	0.105600	0.249853	0.917874	0.804598
5	0.029300	0.237208	0.932367	0.847826
6	**0.005900**	**0.274414**	**0.951691**	**0.901961**
7	0.000500	0.262682	0.932367	0.844444
8	0.001200	0.264597	0.937198	0.857143
9	0.000400	0.271065	0.937198	0.857143
10	0.000300	0.266052	0.937198	0.857143

5.3 Wykres przebiegu treningu



5.4 Wnioski

Wytrenowany model możemy następnie przetestować na nowych danych. W pliku `projekt.ipynb` zawarte są następujące przykładowe wypowiedzi:

1. *"W 2025 roku PKB Polski wzrósł o 10%."*
2. *"Aktualny poziom inflacji wynosi 2,5%."*
3. *"Według mnie to była zła decyzja."*
4. *"Polska znalazła się w gronie 20. największych gospodarek świata.",*
5. *"Uważam, że w XXI wieku edukacja powinna być bezpłatna dla wszystkich."*

Jak można zauważyć, **wypowiedzi nr 1, 2, 4 są niewątpliwie stwierdzeniami i mogą być poddane analizie fact-checkingowej**. Z kolei **zdania nr 3 i 5 to opinie, które nie nadają się do analizy fact-checkingowej**. Wytrenowany model dla każdego z tych zdań zwrócił następujące predykcje:

1. **CLAIM**, Prawdopodobieństwo: 99.22%

2. **CLAIM**, Prawdopodobieństwo: 98.44%
3. **NOT CLAIM**, Prawdopodobieństwo: 100.00%
4. **CLAIM**, Prawdopodobieństwo: 98.83%
5. **NOT CLAIM**, Prawdopodobieństwo: 99.61%

Zatem na powyższym przykładzie widać, że model stosunkowo dobrze radzi sobie z prostymi zdaniami, które można bez wątpliwości zakwalifikować do danej kategorii.

W dalszym etapie, **warto byłoby wykonać bardziej obszerne testy tego modelu**, przykładowo na długim nagraniu zawierającym wywiad z politykiem i z bardziej złożonymi wypowiedziami. Ponadto, **warto byłoby zmodyfikować architekturę modelu, aby użyć modelu językowego, który był trenowany na większej ilości zasobów w języku polskim** niż użyty tutaj SmolLM-1.7B. Dodatkowo, **należałoby rozważyć zamianę klasyfikacji sekwencji na klasyfikację tokenów**, ponieważ wówczas model umożliwiałby ekstrakcję stwierdzeń z dłuższych wypowiedzi - zamiast klasyfikować wybrany fragment wypowiedzi, mógłby znajdować wszystkie weryfikowalne stwierdzenia.

6 Źródła

Do napisania tego opracowania wykorzystałem informacje z poniższych źródeł:

- Materiały do wykładu z Uczenia Maszynowego
- https://d2l.ai
- https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Guide-to-Unlocking-the-Power-of-Sequence-Classification–VmlldzozNDI0NDE4
- https://aclanthology.org/2021.germeval-1.11.pdf
- https://par.nsf.gov/servlets/purl/10166358
- https://zenodo.org/records/3836810
- https://www.geeksforgeeks.org/deep-learning/categorical-cross-entropy-in-multi-class-classification/
- https://www.datacamp.com/tutorial/adamw-optimizer-in-pytorch