Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Sprawozdanie z realizacji projektu z przedmiotu LLM

Porównanie jakości klasyfikacji wieloklasowej modelu językowego klasy tylko-koder z klasyfikatorami wykorzystującymi wektoryzację TF-IDF

Adam Kraś
Numer albumu 325177

WARSZAWA 2025

Spis treści

1. Wstęp

Celem zadania było przeprowadzenie eksperymentów pozwalających na porównanie jakości klasyfikacji pomiędzy dotrenowanym modelem językowym klasy tylko-koder, a "klasycznymi"klasyfikatorami dostępnymi w bibliotece scikit-learn i uzyskanie odpowiedzi na pytanie które z podejść do klasyfikacji tego typu zbioru danych jest "lepsze".

Sam pretrenowany model (**bert-base-uncased**, 110M parametrów) i zbiór danych (**yahoo-answers-topics**, 1.4M próbek) pobrałem z serwisu HuggingFace.

W ramach projektu dotrenowałem model BERT na wybranym przeze mnie zbiorze danych, po uprzednim przeanalizowaniu go. Analiza zawierała zbadanie rozkładu klas i ilościowy podział pomiędzy dane treningowe i testowe. Jako że stosunek wielkości tych zbiorów był daleki od idealnego (i brakowało typowego zbioru testowego) ręcznie zmieniłem ich rozkład.

Przygotowanie danych do skorzystania z gotowych klasyfikatorów z biblioteki scikit-learn wykonałem z identyczny sposób, z tą jedynie różnicą, że najpierw konieczna była wektoryzacja danych TF-IDF.

Wykorzystany przeze mnie zbiór danych zawiera pytania z jednego z forów Yahoo wraz z odpowiedziami i przypisanymi kategoriami (kategorii jest 10). W ramach zadania klasyfikacji przewiduję właśnie kategorię pytania. Każda próbka składa się z następujących atrybutów:

- id - numer identyfikacyjny próbki
- topic - numerycznie zakodowana kategoria pytania
- question_title - tytuł wpisu zawierającego pytanie
- question _content - rozwinięta treść pytania
- best _answer - najlepsza odpowiedź z forum

Do tokenizacji, a co za tym idzie - do dotrenowania modelu - wykorzystałem tylko atrybuty "question _title"i "question _content"ze względu ich informatywność i ograniczone zasoby obliczeniowe. Finalny zbiór danych miał postać "*question _title + question _content → topic*"

Przykładowe próbki ze zbioru danych:

1. Do you stalk someone on Y! Answer, vice verse? why? → *Entertainment & Music*
2. What will you do if you are worried? I'm feeling worried today... please answer my question immediately... T.Y. → *Business & Finance*
3. How are adult stem cells created? → *Science & Education*
4. Why is it considered unlucky to open an umbrella indoors? → *Society & Culture*
5. I'm tired. It's 9:02pm. Should I go to bed or stay up awhile? → *Health*

2. Przegląd literatury

W ramach realizacji projektu korzystałem jedynie z najróżniejszych dokumentacji wykorzystanych narzędzi i wykładów z przedmiotu LLM.

- HuggingFace - biblioteka zawierająca narzędzia wykorzystane do dostrajania modeli oraz sam pretrenowany model i zbiór danych
- PyTorch - biblioteka zawierająca narzędzia umożliwiające wykonywanie obliczeń na karcie graficznej
- Scikit-Learn - biblioteka z gotowymi klasyfikatorami wykorzystanymi do eksperymentów porównawczych
- Matplotlib - biblioteka wykorzystana do generowania wykresów, w szczególności macierzy pomyłek

3. Opis rozwiązania

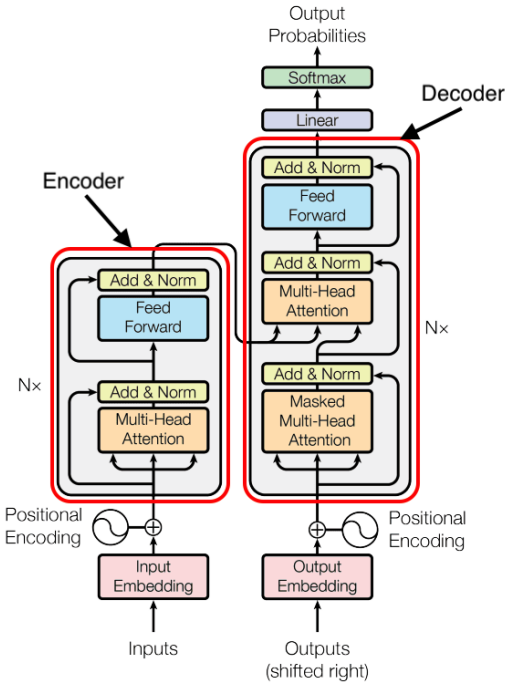
Wykorzystany zbiór danych - yahoo-answers-topic - z serwisu HuggingFace ma 1.4 miliona próbek (opis atrybutów w sekcji Wstęp). W trakcie analizy zbioru danych zauważyłem, że wyjściowy podział zbioru na dane treningowe i testowe był daleki od idealnego (95,9% do 4.1%) oraz, że brakowało typowego zbioru testowego. Z tego względu ręcznie podzieliłem treningowy zbiór danych na nowo w celu otrzymania stosunku rozmiaru zbiorów treningowego, walidacyjnego i testowego 80/10/10. Wszystkie klasy w zbiorze danych były idealnie zbalansowane, zarówno przed, jak i po wprowadzeniu zmian w rozkładzie danych.

Kod do przeprowadzanych eksperymentów napisałem w języku Python, w dużej mierze opiera się on na bibliotece PyTorch. PyTorch jest jedną z najpopularniejszych i najbardziej przystępnych bibliotek umożliwiających łatwe optymalizowanie i dostrajanie treningu modeli na karcie graficznej.

Dostrajanie modelu wykonywałem na własnym sprzęcie z GPU Nvidia RTX 4060 posiadającą 8GB pamięci VRAM. Jest to bardzo wydajna karta, lecz jak dobrze wiemy, zupełnie nieporównywalna z zasobami obliczeniowymi wymaganymi do trenowania wielkich modeli językowych. Z tego względu do projektu zdecydowałem się skorzystać ze względnie małego modelu, tak aby możliwe było jak najdokładniejsze dostrojenie.

Do treningu skorzystałem z narzędzia *Trainer* z biblioteki HuggingFace umożliwiającego łatwe sterowanie hiperparametrami treningu i automatycznie synchronizującego wyniki z serwisem Weights&Biases. Wszystkie hiperparametry i kryteria ustawia się wówczas z wykorzystaniem obiektu *TrainerArguments*.

Model BERT jest modelem językowym klasy tylko-koder. Takie modele dużo lepiej nadają się do zadań klasyfikacji, ponieważ biorą pod uwagę cały kontekst, co umożliwia im tworzenie rozbudowanych reprezentacji sekwencji wejściowych, dzięki czemu lepiej "rozumieją"semantykę klasyfikowanych tekstów. Są one również bardziej efektywne obliczeniowo do tego typu zadań, gdyż nie opierają się na generowaniu kolejnego tokenu.



Rysunek 3.1. Schemat modelu transformer klasy tylko-koder (po lewej)

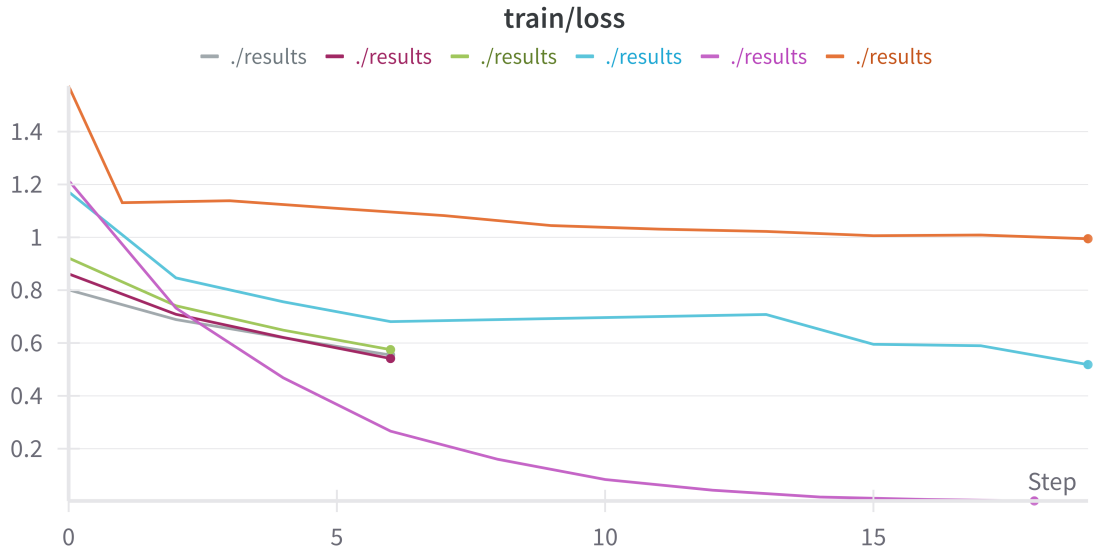
Parametry modelu BERT (110 milionów) zajmowały około 400MB po zaczytaniu do pamięci. Taki rozmiar modelu umożliwił mi zastosowanie podejścia full fine-tuning tzn. dotrenowanie całego modelu na zbiorze danych, a nie tylko kilku warstw najbliżej głowicy klasyfikacyjnej. Mając na uwadze rozmiar zbioru danych i ich złożoność (kilka nieoczywistych przykładów wypisałem w sekcji Wstęp) dostrojenie całego modelu do danych z yahoo-answers-topic wydało się być najlepszym pomysłem. Sprawdziłem również eksperymentalnie wyniki z zamrażaniem warstw i bez, więcej o tym w sekcji Wyniki ewaluacji eksperymentalnej

Do analizy jakości klasyfikacji modelu korzystałem głównie z metryki *accuracy*, obliczającej odsetek poprawnie zaklasyfikowanych przykładów w całym zbiorze testowym. W projekcie wykorzystałem implementację z biblioteki HuggingFace evaluate. Wartość accuracy określa, jaka część etykiet została poprawnie przewidziana przez model, i służyła jako główna miara porównawcza między klasyfikatorami (sklearn vs. BERT).

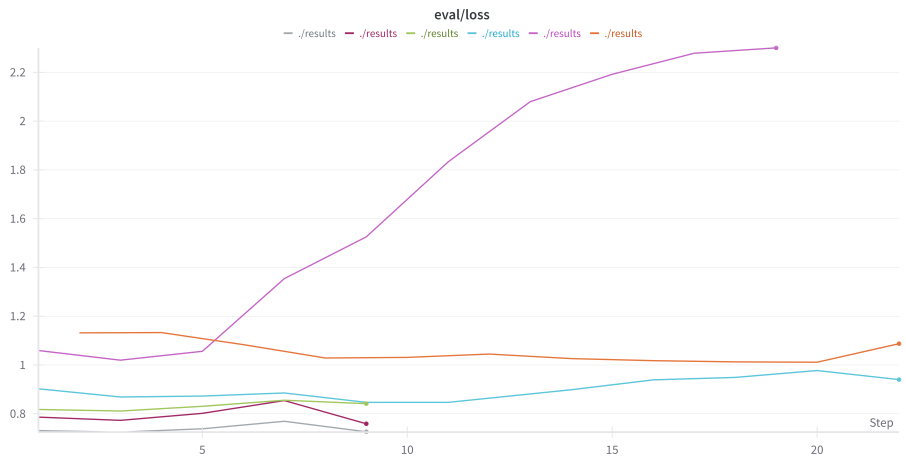
Jak widzimy na poniższych rysunkach, wyniki które otrzymywałem mocno się wahały. Szczególny wpływ na jakość dostrajanego modelu miał rozmiar zbioru danych (pełny czy zredukowany) i ilość dostrajanych warstw. Mimo że model BERT jest stosunkowo mały jak na model językowy, to i tak posiada ponad 100 milionów parametrów, przez co, nawet na pełnym zbiorze danych, bardzo szybko się przeuczał. Kluczowe było zatem dobranie odpowiedniego parametru kroku, rozmiaru wsadu i - przede wszystkim - zaimplementowanie kryterium stopu.

Najlepsze otrzymane przeze mnie wyniki widoczne są na wykresach jako kolor szary

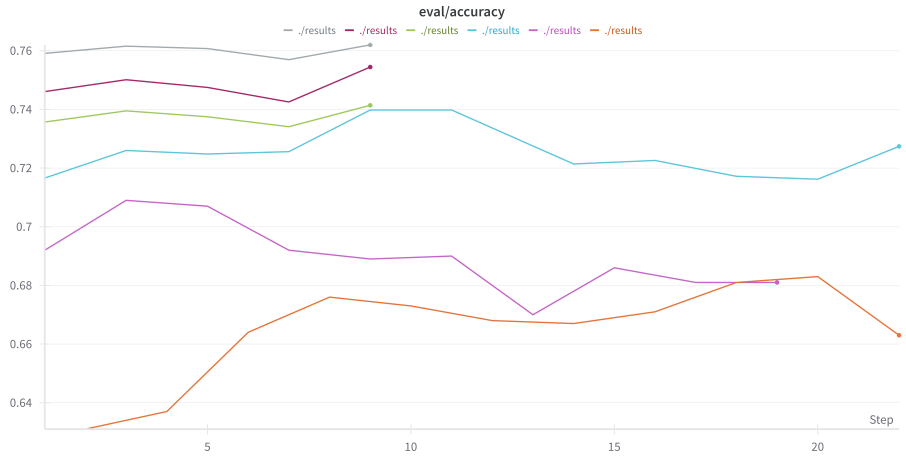
i bordowy - w obu przypadkach zadziałało kryterium stopu, gdyż model przestawał się uczyć, nawet z wykorzystaniem pełnego zbioru danych (1.4 miliona próbek).



Rysunek 3.2. Wartości funkcji straty na zbiorze treningowym modelu BERT



(a) Wartość funkcji straty



(b) Wartość metryki accuracy

Rysunek 3.3. Krzywe uczenia modelu BERT (zbiór walidacyjny)

Do sprawdzenia jakości klasyfikacji gotowych klasyfikatorów (zdecydowałem się sprawdzić model LogisticRegression i OneVsRestClassifier wykorzystujący LogisticRegression) wykorzystałem wektoryzację TF-IDF. Wektoryzacja TF-IDF to metoda która przypisuje każdemu słowu wagę na podstawie jego częstości w dokumencie i rzadkości w całym zbiorze. Dodatkowo TF-IDF umożliwia reprezentację tekstu w przestrzeni cech, co pozwala klasyfikatorom dobrze oddzielić klasy w przestrzeni. Jest to kluczowe, szczególnie w przypadku klasyfikacji wieloklasowej (10 klas).

4. Wyniki ewaluacji eksperymentalnej

Jako że czas treningu modelu BERT był bardzo długi, na początku, do eksperymentalnego dobierania hiperparametrów, zdecydowałem się na dwa zabiegi czasowo usprawniające ten proces:

- **zmiejszanie zbioru danych** - sztucznie zmniejszałem zbiór danych (z zachowaniem balansu klas) co efektywnie sprawiało, że model miał znacząco mniej danych do przetworzenia i mogłem szybciej zobaczyć czy i które zmiany hiperparametrów dają poprawę jakości
- **zamrażanie warstw modelu** - zamrażałem "najniższe"warstwy modelu, czyli te najbardziej oddalone od głowicy klasyfikacyjnej, co również pozwalało dużo szybciej zauważyć czy następuje poprawa w procesie dostrajania

Tak jak wspomniałem w powyższym punkcie, główną metryką służącą do porównywania jakości modeli było *accuracy*, czyli procentowy stosunek poprawnie zaklasyfikowanych przykładów.

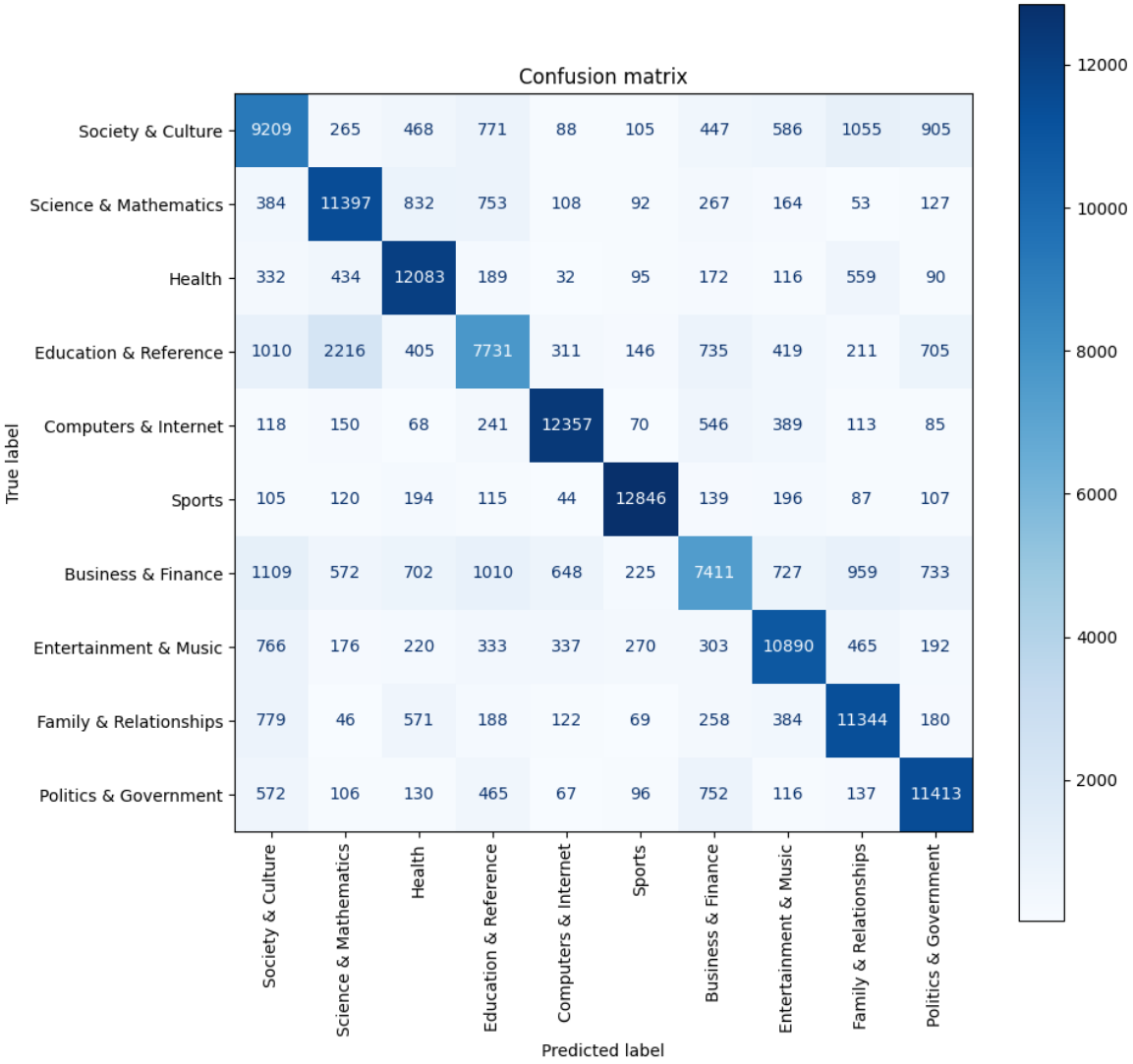
Największym napotkanym problemem było przeuczanie się modelu, który, ze względu na bardzo dużą ilość parametrów szybko "zapamiętywał"próbki ze zbioru treningowego, a dokładność na zbiorach walidacyjnym i testowym spadała.

Najwyższy wynik dla modelu BERT, czyli **76.2%** dokładności na zbiorze testowym udało mi się osiągnąć trenując model przez 4 epoki (kryterium early stopping) na pełnym zbiorze danych i bez zamrażania żadnych warstw modelu (pełny fine-tuning). Wykorzystałem również akumulację gradientu do efektywnego zmaksymalizowania rozmiaru wsadu i regularyzację L2, czyli ustawienie parametru weight_decay w obiekcie klasy *TrainerArguments* w celu jak najskuteczniejszego zapobiegania przeuczaniu.

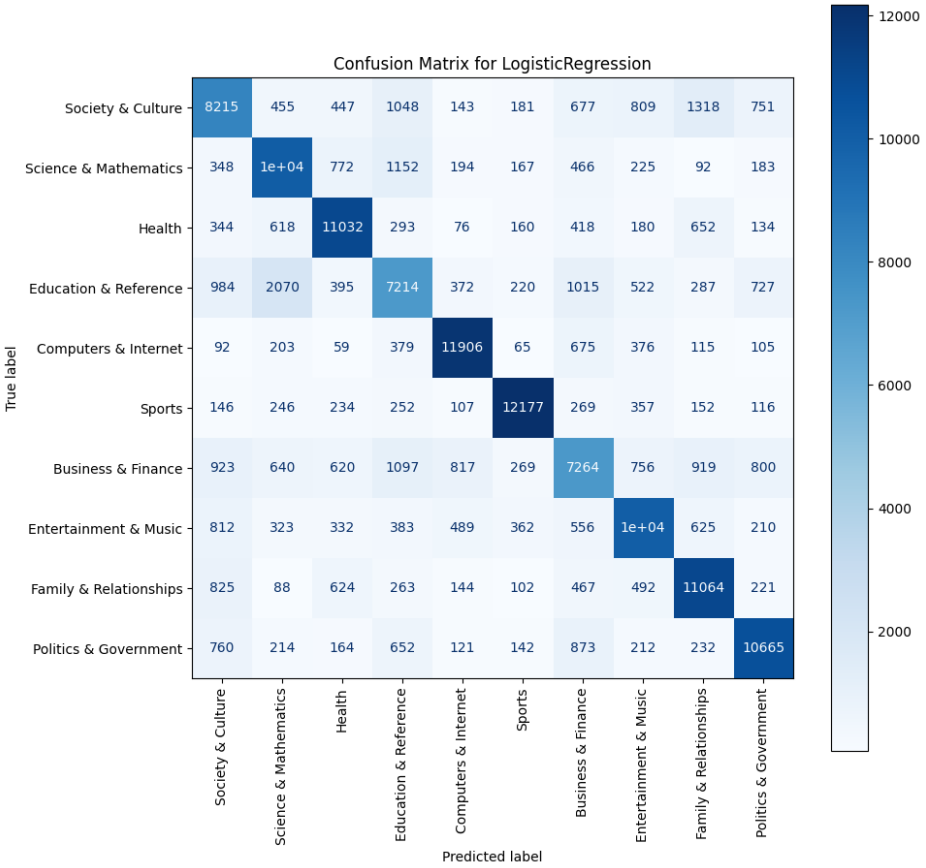
Najwyższe wyniki dla klasyfikatorów z biblioteki sklearn to odpowiednio **71.1%** dla LogisticRegression i **71%** dla OneVsRestClassifier (wykorzystującego również LogisticRegression). Co może się wydawać zaskakujące, OneVsRestClassifier, który tworzy oddzielny klasyfikator binarny LogisticRegression dla każdej klasy, osiągnał niższy wynik niż zwykły, wielomianowy LogisticRegression. Stało się tak zapewne dlatego, że klasyfikatory binarne, w tym przypadku trenowane niezależnie od siebie mogą być mniej skoordynowane i gorzej radzić sobie z klasami podobnymi do siebie. Głównym parametrem do dostosowania w przypadku tych klasyfikatorów był vocab_size, czyli rozmiar słownika. Ostatecznie najlepsze wyniki osiągnąłem dla słownika trzykrotnie większego niż w modelu BERT.

Model BERT osiągał generalnie nieco lepsze wyniki, co wynika głównie z faktu że nauczony jest rozumienia kontekstu, dzięki czemu lepiej rozróżniał podobne klasy od siebie. Co więcej, korzysta z pretrenowanych wag, zawierających ogromną wiedzę językową wyciągniętą z dużych zbiorów.

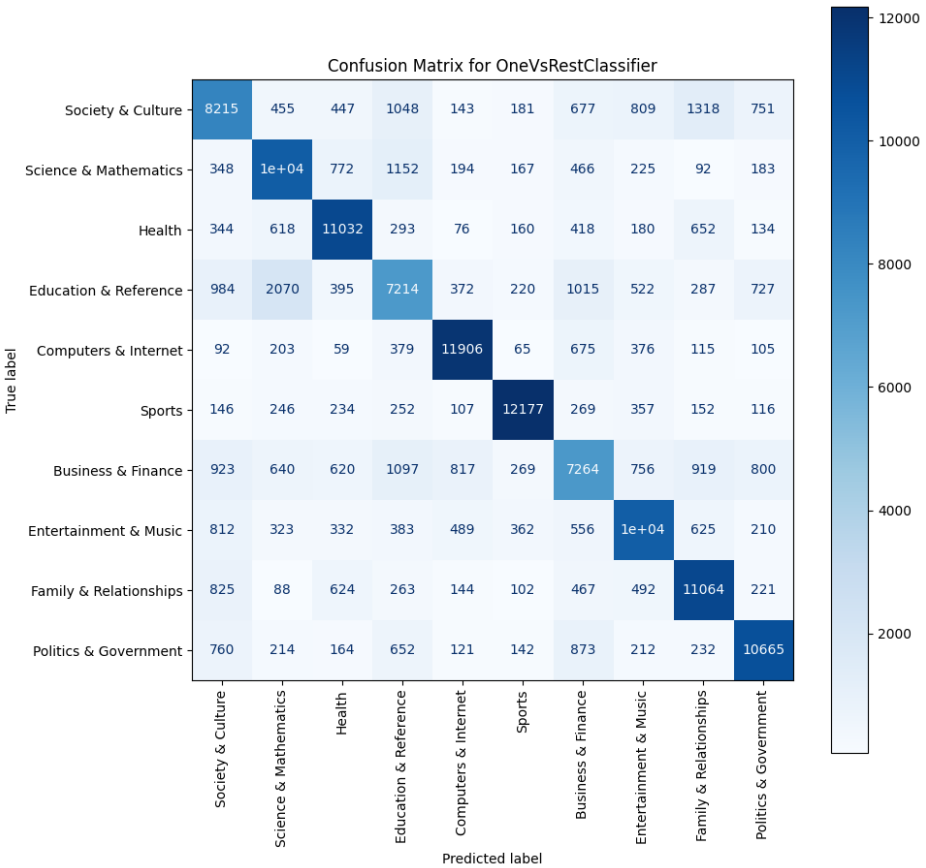
W zbiorze danych występowało parę klas szczególnie problematycznych (podobnych do siebie), co dobrze widać na poniższych macierzach pomyłek.



Rysunek 4.1. Macierz pomyłek dla modelu BERT



(a) LogisticRegression



(b) OneVsRestClassifier+LogisticRegression

Rysunek 4.2. Macierze pomyłek dla klasyfikatorów z sklearn

Jak widzimy, dla wszystkich podejść zdecydowanie najwięcej problemów sprawiło rozróżnienie klasy *Science&Mathematics* i *Education&Reference*. Dodatkowo, generalnie najgorzej rozpoznawanymi klasami były *Education&Reference* oraz *Business&Finance*. Nie jest to szczególnie zaskakujące, mając na uwadze jak niewielkie semantyczne różnice pomiędzy tymi kategoriami mogą się pojawiać. W przypadku każdego z modeli, najlepiej rozpoznawanymi klasami były *Computers&Internet* oraz *Sports*. Model BERT bardzo dobrze poradził sobie również z klasą *Health*. Każda z tych trzech to klasy, które dość trudno semantycznie powiązać z jakąkolwiek inną w zbiorze danych, zatem tu też wynik jest spodziewany.

5. Podsumowanie

W przeprowadzonych eksperymentach model BERT osiągnął wyższą dokładność, lecz wiązało się to z dużo dłuższym czasem treningu - dla najlepiej dostrojonego modelu około 5.5 godziny. Osiągnięta najwyższa dokładność - ponad 76% jest blisko najwyższych wyników osiąganych przez użytkowników HuggingFace na tym zbiorze danych (wynoszących około 78-79%). Rezultat ten zatem uważam za mały sukces.

Porównanie jakości modelu językowego z gotowymi klasyfikatorami również nie przyniosło zaskakujących rezultatów. Model językowy potrafił lepiej dostosować się do tak skomplikowanego problemu, co z drugiej strony niestety wiązało się z wielokrotnie większymi wymaganiami w kwestii czasu i zasobów obliczeniowych. Dobór "lepszego"rozwiązania będzie zatem dla każdego kwestią indywidualną, konieczna jest tu odpowiedź na pytanie *Czy jestem w stanie poświęcić wielokrotnie więcej zasobów dla osiągnięcia nieco lepszej jakości klasyfikacji?*.

Prawdopodobnie klasyfikatory z biblioteki sklearn osiągnęłyby lepsze wyniki gdyby wykorzystać bardziej zaawansowane reprezentacje tekstu takie jak np. word embeddings.

Nasz dostrojony model językowy najpewniej można by zamienić na jeszcze mniejszy (typu DistilBERT lub TinyBERT) bez znaczących strat w jakości klasyfikacji, ale za to ze skróconym czasem dostrajania.