Architektura PRADO i jej modyfikacje w zadaniu badanie wydźwięku wypowiedzi

(PRADO architecture and its modifications in sentiment analysis)

Oskar Bujacz

Praca inżynierska

Promotor: dr Paweł Rychlikowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

5 lutego 2022

Streszczenie

Zadanie wykrywania mowy nienawiści jest już dobrze zbadaną dziedziną dla języka angielskiego, ale w przypadku języka polskiego jest to wciąż obszar ciągłych badań. Jest to jeden z powodów zawarcia tego zadania w konkursie PolEval 2019. W pracy proponujemy nietrywialną modyfikację nowatorskiej sieci neuronowej PRADO dedykowanej do zadania klasyfikacji tekstów – zamiast losowej trenowalnej projekcji stosujemy klasyczne zanurzenia słów. Porównujemy różne warianty zanurzeń takie jak word2vec, FastText czy też modyfikacje Localilty-Sensitive Hashing. Podajemy też różne możliwości wstępnego przetworzenia danych tekstowych. Pokażemy, że zastosowanie zanurzeń pozwala osiągnąć istotnie lepsze wyniki nie tylko od standardowej implementacji z projekcją, ale również od pozostałych uczestników wspomnianego konkursu PolEval.

Hate speech detection task is already a well-researched field for the English language, but in the case of the Polish language it is still an area of ongoing research. This is one of the reasons for including this task in the PolEval 2019 competition. In the thesis, we propose a non-trivial modification of the state-of-the-art PRADO neural network created for the task of text classification - instead of a random trainable projection, we use classic word embeddings. We compare different kinds of embeddings such as word2vec, FastText or Localilty-Sensitive Hashing modifications. We also provide various options for data preprocessing. We will show that the use of embeddings allows to achieve significantly better results not only than the standard implementation with projection, but also than other participants of the PolEval competition.

Spis treści

Rozdział 1.

Wprowadzenie

1.1. Zadania przetwarzania języka naturalnego

Przetwarzanie języka naturalnego (Natural Language Processing, NLP) ma swoje korzenie już w latach 50 XX wieku, kiedy to Alan Turing tworząc podwaliny pod sztuczną inteligencję zaproponował test Turinga. Poprzez rozmowę z programem komputerowym użytkownik miał stwierdzić, czy rozmawia z człowiekiem, czy tylko z jego imitacją. Stąd wywodzi się pierwotna nazwa tego testu, czyli *imitation game* [1].

NLP to dziedzina na pograniczu językoznawstwa, informatyki i sztucznej inteligencji opisująca proces analizy, rozumienia i tłumaczenia języka naturalnego przez komputer. Obejmuje ona wiele różnych zadań i problemów. Jednymi ze znanych wyzwań są problemy klasyfikacji całych tekstów (analiza wydźwięku, ang. sentiment analysis) lub tylko ich części (Named Entity Recognition, Part-Of-Speech tagging), tworzenie streszczeń, detekcja fake news, tłumaczenie maszynowe czy też tworzenie czatbotów.

Wraz ze wzrostem ilości dostępnych danych oraz mocy obliczeniowej zmieniały się podejścia do powyższych problemów proponowane przez badaczy. Do lat 90 XX wieku głównym rozwiązaniem było stosowanie sztywnie zdefiniowanych reguł i zasad, z wykorzystaniem wiedzy eksperckiej. Niestety zdefiniowanie wszystkich sytuacji w języku naturalnych jest niemożliwe, ze względu na rozmiar problemu i ciągłą ewolucję mowy i pisma. Przełomowym wydarzeniem było rozpowszechnienie się metod statystycznych. Korzystały one z rosnącej ilości dostępnych danych oraz z ówcześnie sformułowanych narzędzi i wczesnych modeli uczenia maszynowego, takich jak: neuronowe sieci rekurencyjne [2], pierwsze reprezentacje słów jako wektory liczb [3] czy nieco wcześniej opisane n-gramy [4].

Prawdziwa rewolucja nastąpiła jednak dopiero w latach 2000, kiedy to zaczęto szerzej wykorzystać znane już wcześniej głębokie sieci neuronowe [5]. Chociaż pierwszy neuronowy model językowy powstał już w 2003 roku [6], to dopiero rok 2012

można uznać za początek wciąż trwającej rewolucji głębokiego uczenia. Właśnie wtedy konkurs ImageNet wygrała ze znaczną przewagą splotowa sieć neuronowa AlexNet [7].https://www.overleaf.com/project/61e1827dc4ee9f24c3c90b45

1.2. Wykrywanie nienawiści

Wykładniczy wzrost popularności mediów społecznościowych zrewolucjonizował komunikację międzyludzką i sposób publikowanie treści, ale jednocześnie znacznie ułatwił szerzenie się nienawiści. Anonimowość, brak bezpośredniego kontaktu z rozmówcą oraz poczucie braku konsekwencji są głównymi powodami występującego problemu. Wykrywanie nienawiści (hate detection [1]) jest sposobem na poradzenie sobie z tym problemem. Jest to jedno z zadań klasyfikacji tekstu z dziedziny analizy wydźwięku wypowiedzi.

Formalnie mowę nienawiści możemy zdefiniować jako: „*ogół wypowiedzi zawierających elementy wyszydzające, poniżające osobę lub grupy osób ze względu na ich cechy takie, jak np. płeć, kolor skóry, wyznanie, niepełnosprawność*" [8]

Z powyższej definicji można wywnioskować kilka możliwych podejść do zadania *hate detection*. Prostsze rozwiązanie to klasyfikacja binarna, kiedy to algorytm decyduje, czy wypowiedź jest nacechowana negatywnie. Trudniejszą propozycją jest podział tekstów na więcej niż dwie klasy, w celu wykrywania wypowiedzi z konkretnej kategorii mowy nienawiści. Przez lata wielkie korporacje stojące za współczesnymi sieciami społecznościami takie jak Facebook, Twitter czy Google inwestowały miliony dolarów w celu zwalczania negatywnych wypowiedzi, jednak wciąż były one krytykowane za brak wystarczającej skuteczności. Powodem były naciski na moderację ręczną, która ma wiele wad, m.in. subiektywność i ograniczoną moc przerobową [9]. Dlatego obecnie preferowanych podejściem jest automatyczna klasyfikacja tekstów.

1.3. Przegląd literatury

W literaturze można znaleźć wiele przykładów zadań automatycznej analizy tekstu. Najpopularniejszym przykładem jest analiza wydźwięku wypowiedzi. Do niej należą m.in. wykrywanie mowy nienawiści w sieciach społecznościowych, grupowanie zgłoszeń w dziale wsparcia użytkowników według wagi problemu na podstawie tonu wypowiedzi, wykrywanie recenzji przydatnych dla innych kupujących [10]. Innymi przykładowymi zadaniami klasyfikacji są: przypisanie tekstu do kategorii, np. segregacja wiadomości na stronie internetowej, wykrywanie języka wypowiedzi w

[1] Z uwagi na powszechność angielskich terminów w przetwarzaniu języka naturalnego, będą one pojawiać się w nawiasach.

celu jej późniejszego przetłumaczenia, problem detekcji spamu czy też wykrywanie obecnych trendów poprzez analizę wypowiedzi w sieciach społecznościowych.

Algorytmy stosowane w analizie wydźwięku wypowiedzi możemy podzielić na dwie kategorie: w pierwszej z nich znajdziemy mechanizmy oparte na manualnym przetwarzanie tekstu i utworzeniu z niego cech, które są wejściem dla algorytmu, takie jak SVM (Support-Vector Machines), Naive Bayes, metody związane z drzewami decyzyjnymi, lasami losowymi i regresją logistyczną. Możemy je nazwać algorytmami klasycznymi. Druga kategoria obejmuje głębokie sieci neuronowe, które stosują podejście automatycznego tworzenia użytecznych cech dla modelu [9].

W przypadku algorytmów klasycznych wytworzenie sensownych cech jest kluczowe dla skuteczności, są one też przydatne przy przetwarzaniu danych wejściowych dla sieci neuronowej. Jako proste cechy możemy przyjąć techniki bag-of-words oraz n-gramy na słowach i literach, są to podstawowe cechy dla zadania klasyfikacji nienawiści [9]. Warto również zwrócić uwagę na adresy URL, wzmianki o innych osobach/organizacjach, hasztagi, interpunkcję, długość tekstu, liczbę i rodzaj emotikon, liczbę wielkich liter w tekście jak i wiele innych.

Osobną kategorią są cechy lingwistyczne, takie jak używane części mowy i drzewa zależności w zdaniu. Ciekawym podejściem jest skorzystanie z danych o użytkowniku stojącym za daną wypowiedzią, takie jak jest poprzednia aktywność, jego indywidualny styl wypowiedzi i rodzaje postów.

W kategorii głębokiego uczenia szczególną popularnością wyróżnia się kilka architektur. Do ekstrakcji cech z tekstu szeroko wykorzystywane są splotowe sieci neuronowe (Convolutional Neural Networks, CNN) [11]. Pozwalają one automatycznie wykryć przydatne cechy dla konkretnego zadania. Intuicyjnie uczą się cech podobnych np. do n-gramów Do samej klasyfikacji wypowiedzi stosowane są rekurencyjne sieci neuronowe (Recurrent Neural Network, RNN), szczególnie wyspecjalizowane architektury takie jak GRU (Gated Recurrent Network) i LSTM (Long Short-Term Memory Network) [12]. Ten typ sieci jest wybierany z powodu sekwencyjnej natury ludzkiej wypowiedzi. W zadaniu wykrywania nienawiści możemy też spotkać użycie obecnie najbardziej zaawansowanych modeli językowych, architektur typu transformer korzystających z mechanizmu uwagi [13].

Ważnym elementem badań w NLP jest obiektywna ocena kilku różnych podejść do tego samego problemu. W tym celu dla języka angielskiego powstał benchmark GLUE (The General Language Understanding Evaluation) [14] oraz jego późniejsza ewolucja SuperGLUE [15]. Wzorem tej inicjatywy dla języka polskiego również został utworzony benchmark do ewaluacji różnych systemów z zakresu NLP - KLEJ (Kompleksowa Lista Ewaluacji Językowych) opracowany przez firmę Allegro [16]. Jest to zbiór dziewięciu zadań ewaluujących modele językowe dla języka polskiego. Pojawiają się tam zadania dotyczące analizy wydźwięku recenzji produktów, analizy streszczeń artykułów czy też zadanie typu Named Entity Recognition. Co ciekawe, rozważane w pracy zadanie wykrywanie mowy nienawiści znalazło się również w

konkursie PolEval. Pokazuje to duże znaczenie tego problemu.

Zadanie wykrywania nienawiści w przypadku języka polskiego wciąż nie jest dobrze zbadanych tematem [17], szczególnie w porównaniu do języka angielskiego. Dodatkowo istnieje bardzo mała liczba przygotowanych danych z adnotacjami do trenowanie modeli [18].

1.4. Konkurs PolEval

Konkurs PolEval to inicjatywa utworzona przez Instytut Informatyki Polskiej Akademii Nauk (PAN) w 2017 roku, która skupia się na przetwarzaniu języka naturalnego, promując badania w tym zakresie na przykładzie języka polskiego [17]. Motywacją dla tego przedsięwzięcia była potrzeba obiektywnej oceny różnych podejść do tego samego zadania oraz rosnące zainteresowanie dziedziną NLP. Obecnie trwa zgłaszanie zadań do edycji 2022.

PolEval to otwarty konkurs, w którym uczestnicy mogą zmierzyć się z różnymi zadaniami z zakresu przetwarzania języka naturalnego i rozumienia języka naturalnego (Natural Language Understanding, NLU). Inspiracjami dla tej inicjatywy były podobne przedsięwzięcia dla innych języków, takie jak SemEval [19] (różne języki) czy Evalita (język włoski) [20].

Przykładowo, w roku 2019 PolEval zaproponował następujące zadania. Oryginalne nazwy zostały podane w języku angielskim.

1. Task 1: Recognition and normalization of temporal expressions
   Zadanie 1: Rozpoznawanie i normalizacja wyrażeń czasowych

2. Task 2: Lemmatization of proper names and multi-word phrases
   Zadanie 1: Lematyzacja nazw własnych i fraz składających się z wielu słów

3. Task 3: Entity linking
   Zadanie 3: Dodawanie odnośników do fraz w tekście

4. Task 4: Machine translation (EN-PL, PL-RU, RU-PL)
   Zadanie 4: Tłumaczenie maszynowe

5. Task 5: Automatic speech recognition
   Zadanie 5: Automatyczne rozpoznawanie mowy

6. Task 6: Automatic cyberbullying detection (harmful vs non-harmful and detecting type of harmfulness)
   Zadanie 6: Automatyczne wykrywanie cyberprzemocy (nienawiść i normalne wypowiedzi oraz różne typy nienawiści)

Zadanie z konkursu PolEval 2019, które na którym się skupimy to Task 6: Automatic cyberbullying detection [17]. Ten problem był zdecydowanie najbardziej popularny wśród uczestników. Zaproponowano 16 podejść, kiedy do pozostałych pięciu zadań było ich 18. Liczba rozwiązań pozwala sądzić, że wyniki są bliższe optymalnym niż w pozostałych zadań. Dzięki temu późniejsze porównanie otrzymanych wyników do wyników zawodników jest bardziej obiektywne.

1.5. Wybrane zadanie 6

Zadanie 6-te składa się z dwóch podpunktów: podpunkcie 6.1 uczestnicy mieli przypisać tweety do dwóch klas: wypowiedź nacechowana nienawistnie (harmful/hate) lub bez negatywnego wydźwięku (non-harmful/ normal). W podpunkcie 6.2 klasa nienawiści została dodatkowa rozdzielona na dwie podklasy: cyberprzemoc (cyberbullying) oraz mowa nienawiści (hate-speech). Kluczowa różnica między tymi pojęciami jest następująca: cyberprzemoc jest skierowana do jednostek, natomiast hate-speech jest zaadresowany do grupy osób [21].

1.6. Cel i zakres pracy

W przedstawionej pracy skupiono się na zadaniu 6.1. Celem pracy jest sprawdzenie efektywności nowatorskiej architektury PRADO zaprezentowanej w 2019 przez pracowników Google tuż po ogłoszeniu wyników PolEval 2019.

Początkowo pokażemy jak dobre wyniki wyniki można osiągnąć z podstawowymi algorytmami uczenia maszynowego takimi jak Naiwny Klasyfikator Bayesowski, drzewa decyzyjne czy Support Vector Machines, celem wyznaczenia pewnego akceptowalnego dolnego progu dla wyników . Później przetestujemy oryginalną sieć PRADO bez żadnych modyfikacji. Następnie, dodamy do niej nietrywialną modyfikacją - zamiast stosowanej projekcji pozwalającej znacznie zmniejszyć jej rozmiar, zastosujemy klasyczne zanurzenia słów. Kosztem wielkości modelu, pokażemy w jakim stopniu możemy poprawić wyniki w zadaniu w stosunku do oryginalnej implementacji. Rozważymy również różne warianty zanurzeń wraz z różnorodnym wstępnym przetworzeniem korpusu tekstów przed utworzeniem wektorowej reprezentacji słów z tego zbioru.

Odniesiemy się również do wyników otrzymanych przez uczestników konkursu PolEval, pokazując, że możemy uzyskać lepsze wyniki od każdej z zaprezentowanych metod.

Rozdział 2.

Sieć neuronowa PRADO - opis architektury

2.1. Motywacja obliczeń bezpośrednio na urządzeniu

Badania w zakresie zadania klasyfikacji tekstów przy ograniczonych zasobach doprowadziły do zaproponowania architektury PRADO (Projection Attention Networks for Document Classification On-Device) [22]. Jest to nowatorska sieć neuronowa korzystająca z wielu ostatnich osiągnięć badaczy, takich jak mechanizm uwagi [23] czy splotowe sieci neuronowe [24]. Sieć została zbudowana z zamiarem uruchamiania jej na urządzeniach o małej mocy obliczeniowej i niewielkiej ilości dostępnej pamięci. Ostatnie prace pokazują, dlaczego obliczenia wykonywane bezpośrednio na urządzeniu użytkownika końcowego są kluczowe dla zachowanie prywatności oraz zwiększenia komfortu użytkowania [25, 26].

Głównym problemem takiego podejścia są ograniczone zasoby w porównaniu do podejść korzystających z rozwiązań chmurowych. Nie wszystkie architektury mogą być uruchomione na współczesnym telefonie komórkowym, ponadto chcielibyśmy wciąż otrzymywać zbliżone wyniki do tych z większych, bardziej wysublimowanych sieci. Zaproponowano rozwiązanie tego problemu bazujące na sieci typu self-governing (self-governing neural network, SGNN) [27] opartych na lokalnie wrażliwych projekcjach [25, 26]. Rozważana architektura PRADO idzie o krok dalej i rozwija pomysł statycznych projekcji w SGNN, poprzez uczynienie ich trenowalnymi. Ponadto dodaje mechanizm uwagi wraz ze splotem, co pozwala na zauważenie przez sieć daleko położonych zależności w klasyfikowanych tekstach. Rysunek 2.2 przedstawia schemat PRADO. Składa się ona z kilku warstw. Na początku tokeny trafiają do projekcyjnej warstwy zanurzającej, następnie są one przetwarzane przez sieć splotową i enkoder z mechanizmem uwagi, a na koniec znajduje się klasyczna w pełni połączona sieć klasyfikująca.

2.2. Warstwa projekcji

Na początku załóżmy, że chcemy znaleźć reprezentację wektorową dla słowa w_i znajdującego się w tekście o T tokenach. Przyjmując liczbę unikalnych słów w korpusie jako V, słowo w_i reprezentujemy jako $\delta_i \in R^V$ (delta Diraca). Następnie, w przypadku klasycznych zanurzeń skorzystamy z macierzy wag $W \in R^{d \times V}$, gdzie d to wcześniej zdefiniowana długość wektorów. Wtedy szukany wektor słowa to $e_i = W\delta_i$. Macierz W w przypadku dużego korpus może zawierać wiele milionów słów, co powoduje duże wymagania pamięci w celu ich przechowywania. Dodatkowo zakładamy, że znamy tylko V słów, co utrudnia znalezienie słów nie znajdujących się w korpusie [28, 29]

Podejście w PRADO zastępuje wspomniane zanurzenia poprzez projekcję. Zamiast stosowania mapowania w_i do δ_i, korzystamy z operatora projekcji, aby uzyskać f_i. Poprzednie prace [25, 27, 26] pokazały, że metody wykorzystujące projekcję mogą pomóc utworzyć niewielkie sieci neuronowe, które wciąż są w stanie osiągać dobre rezultaty. Innowacyjną częścią jest udoskonalenie procesu projekcji, poprzez uczynienie jej *trenowalną*. Dodatkowo, nie jest ona oparta na uprzednio określonym zbiorze słów.

W celu utworzenia zastąpienia zanurzeń, najpierw pojedynczy token w_i jest kodowany za pomocą $2B$ bitów za pomocą pewnej funkcji haszującej. Następnie, operator P mapujemy każde dwa kolejne bity do wektora $f_i \in \{-1, 0, 1\}^B$. Ciągi bitów `00` i `11` kodowane są na `0`, ciąg `01` na `1`, natomiast ciąg `10` na `-1`. Zakodowanie za pomocą jedynie 3 wartości znacznie przyśpiesza czas treningu i zmniejsza rozmiar w stosunku do wartości typu *float*, a jednocześnie znacznie zwiększa możliwości zapamiętania informacji w stosunku do wektorów binarnych. [30]. Następnie, za pomocą warstwy sieci neuronowej uzyskujemy $e_i = \phi(f_i)$. Zwróćmy uwagę na liczbę parametrów dla funkcji ϕ. Jest ich jedynie $B \cdot d$, gdyż mapujemy B bitowe projekcję na zanurzenia o d wymiarach. W praktyce, $B \in [128, 512]$, $d \in [32, 96]$ zatem $B \ll V$.

2.3. Warstwa splotowa i mechanizm uwagi

Autorzy pracy pokazali enkoder, który mapuje sekwencje projekcji kolejnych słów, $e_0, \ldots, e_{T-1}$, do wektora o ustalonej długości, który reprezentuje cały wejściowy tekst, gdzie T to długość tekstu.

W poprzednich pracą często używano splotu jednowymiarowego, która nie zawsze jest najlepszym wyborem. W zadaniu detekcji mowy nienawiści nie wszystkie słowa są równie ważne dla przewidywanej klasy, dlatego metody takie jak average pooling nie są efektywne. W związku z tym autorzy zaproponowali użycie dwóch niezależnych splotowych sieci neuronowych. Pierwsza z nich to sieć F typu *projected*

features, ma za zadanie znaleźć cechy ważne dla zadania. W tym celu zastosowano splot na każdej z projekcji e_i, gdzie $i \in [0, T-1]$ to indeks słowa tekście, a T to długość tekstu.

$$F_i^n = Conv(e_i, n, N) \tag{2.1}$$

gdzie n to rozmiar filtra splotowego, N to liczba wyjściowych pasm splotu, $F_i^n \in \mathbb{R}^n$. Druga z sieci A to *attention network*, która ma za zadanie wybrać cechy ważne dla klasyfikacji.

$$W_i^n = Conv(e_i, n, N) \tag{2.2}$$

gdzie podobnie jak wcześniej n to rozmiar filtra splotowego, N to liczba wyjściowych pasm splotu, $W_i^n \in \mathbb{R}^n$. Sam splot wygląda bardzo podobnie, ale po nim używamy funkcji softmax. Dzięki temu uzyskujemy rozkład prawdopodobieństwa dla sekwencji słów, który przedstawia cechy ważne dla zadania wykrywania nienawiści.

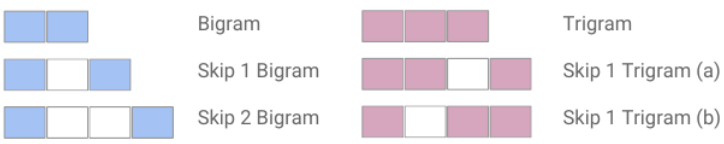
$$A_i^n = \frac{e^{W_i^n}}{\sum_{i=0}^{T-1} e^{W_i^n}} \tag{2.3}$$

Następnie liczymy wartość oczekiwaną F używając rozkładu A otrzymując kodowanie E^n o stałej długości dla pewnego filtra splotowego n. T to długość tekstu. Ta operacja w przypadku rozkładu jednostajnego upraszcza się do average pooling, a w przypadku funkcji zwracającej minimum bądź maksimum otrzymujemy odpowiednio max lub min pooling.

$$\mathcal{E}^n = \sum_{i=0}^{T-1} A_i^n F_i^n \tag{2.4}$$

2.4. Sekwencyjne filtry splotowe

Następnie aplikujemy filtry n-gramowe dla każdego wektora E^n. Rozmiar filtra jest równy n. Poza filtrami n-gramowymi, używane są również filtry splotowe z maską, które mają na celu symulację efektu skip-gramów 2.1. Każdy filtr o rozmiarze n generuje kodowanie E^n tekstu wejściowego



Rysunek 2.1: Schemat symulacji skip-gramów poprzez filtry splotowe z maską [22].

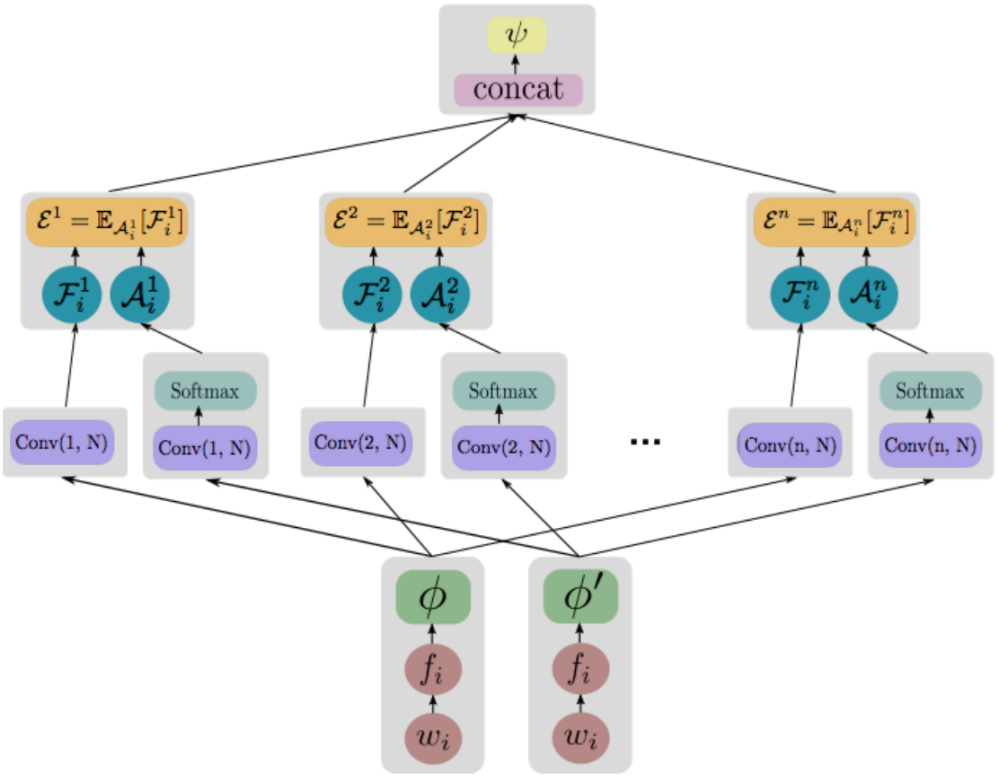
2.5. Warstwa klasyfikująca

Filtr splotowy danej szerokości n pozwala reagować na różne n-gramy na słowach. Otrzymane cechy łączymy w macierz poprzez operację `concat`. Zwracamy reprezentację wejściowego tekstu x_k o ustalonym rozmiarze.

$$TextEncoder(x_k) = concat(\mathcal{E}^1, \mathcal{E}^2, \dots) \tag{2.5}$$

Na końcu stosujemy zwykłą sieć typu *fully-connected feed-forward*, która ostatecznie zwraca wynik klasyfikacji. W zależności od tego, czy problem jest wieloklasowy czy binarny, stosujemy funkcję aktywacyjną typu softmax bądź sigmoid.

$$output = \psi(TextEncoder(x_k)) \tag{2.6}$$



Rysunek 2.2: Architektura PRADO. w_i to wejściowe słowo, f_i to jego projekcja. Za pomocą ϕ otrzymujemy zanurzenia z projekcji. F_i^j i A_i^j to wektory wyjściowe sieci splotowych. $\mathcal{E}^j$ to wartość oczekiwaną F_i^j z rozkładu $E_{A_i^j}$ ψ to wynik klasyfikacji. [22].

Rozdział 3.

Wektorowe reprezentacje słów

3.1. Zanurzenia słów

Rozważania o wektorowej reprezentacji słów warto zacząć od podejścia typu Bag-of-Words, gdzie każde słowo jest uznawane są osobną cechę. Wtedy wektor słowa to $w_i \in \mathbb{R}^n$, gdzie n to liczba różnych słów, a jedyny niezerowy element jest pod indeksem i. Powstaje wtedy rzadka macierz M o bardzo dużym rozmiarze, $M \in \mathbb{R}^{n \times n}$. Dlatego istnieją rozwiązania, które korzystają z dużo mniejszych gęstych macierzy $M' \in \mathbb{R}^{n \times d}$, gdzie $d \ll n$. Kolejne wymiary wektorów nie reprezentują tutaj kolejnych słów, tylko bardziej ogólne cechy języka.

Zanurzenia słów są obecnie bardzo ważnym elementem współczesnego przetwarzania języka naturalnego. Pozwalają one rozszerzyć wciąż stosowany model n-gramowy, używany do statystycznego modelowania języka. Założenie jest następujące: każde słowo w chcemy reprezentować jako pewien wektor $v \in R^d$, gdzie d to wybrany rozmiar zanurzenia. Głównymi zaletami tego podejścia jest możliwość reprezentacji kontekstu słów w tekstach oraz ich semantycznego i syntaktycznego podobieństwa. Oczekujemy, że słowa blisko spokrewnionego będą mieć podobną reprezentację wektorowa np. w[książka] $\approx w$[podręcznik] lub w[gazela] $\approx w$[antylopa]. Jako miary podobieństwa między wektorami możemy stosować odległość euklidesową 3.1 bądź podobieństwo cosinusowe 3.2.

$$S_{Eucl} = d(X, Y) = \left(\sum_{i=1}^{n} (X_i - Y_i)^2 \right)^{\frac{1}{2}} \tag{3.1}$$

$$S_{\cos}(X, Y) = \cos(\theta) = \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2} \sqrt{\sum_{i=1}^{n} Y_i^2}} \tag{3.2}$$

Przed obecnie stosowanymi rozwiązaniami, słowa były kodowane za pomocą metod statystycznych. Przykładem jest algorytm Latent Semantic Indexing [31].

Niestety korzysta ona z rozkładu według wartości osobliwych (Singular Value Decomposition, SVD), co nie pozwala łatwo dodać kolejnego słowa do słownika. Jest ona również czuła na brak balansu w częstotliwości wystąpień poszczególnych słów. Problemem jest również wysoka złożoność obliczeniowa $O(mn^2)$ gdzie m i n to wymiary macierzy współwystąpień.

3.1.1. Word2Vec

Neuronowe modele językowe istniały już od lat 2000 [6], ale pierwsza efektywna propozycja została zaprezentowana dopiero w 2013 roku, szerzej jest znana jako Word2Vec [28]. Mimo, że wcześniej istniały próby rozwiązania tego problemu takie jak, Feedforward Neural Net Language Model (NNLM) [6] oraz Recurrent Neural Net Language Model (RNNLM) [32], nie pozwalały one na trening przydatnych zanurzeń. Autorzy ostatniej pracy utworzyli wektory długości między 50-100 dla jedynie $\approx$ 17 tys. najczęstszych słów, przy korpusie liczącym kilka milionów słów. Warto wspomnieć że ówcześnie trening zajął 3 tygodnie przy 40 jednostkach CPU.

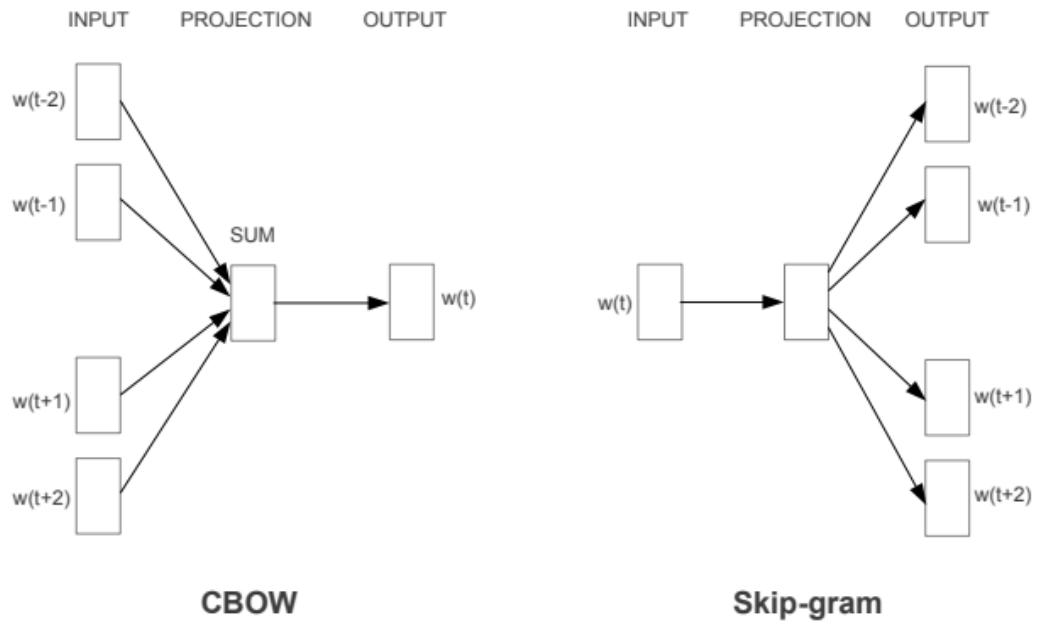
Autorzy architektury Word2Vec zaproponowali dwie istotnie różne architektury - Continuous Bag Of Words (CBOW) oraz Skip-gram. Obydwie korzystają z prostej sieci neuronowej z jedną warstwą ukrytą o N neuronach, gdzie N to pożądany wymiar wektorów zanurzeń. Wektory zanurzeń to macierz wag warstwy ukrytej. W pierwszej metodzie CBOW, kontekst słowa jest wejściem sieci, celem jest przewidzenia aktualnego słowa w tekście. Oczywiście kontekstem może być więcej niż jedno słowo. W przypadku Skip-gram jest odwrotnie, wejście to pojedyncze słowo, wyjście to kontekst słowa. Schemat sieci znajduje się na rysunku 3.1. Poniżej widzimy funkcje celu dla obydwóch architektur.

$$\mathrm{L}_{\mathrm{CBOW}} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log p(w_t|w_{t+j}) \tag{3.3}$$

$$\mathrm{L}_{\mathrm{Skipgram}} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log p(w_{t+j}|w_t) \tag{3.4}$$

gdzie T to liczba unikalnych słów w korpusie, m to długość kontekstu, a

Mimo znaczących możliwości modelu, nie jest on idealny. Nie potrafi utworzyć dwóch różnych wektorów dla słów o tej samej pisowni, ale różnym znaczeniu. Nie radzi sobie również z nieznanymi słowami (Out Of Vocabulary, OOV). Ten pierwszy problem został uwzględniony m.in. w modelu BERT [33] oraz ELMO [34] poprzez dopuszczenie zanurzeń kontekstowych. Ten drugi został rozwiązany w kolejnej opisywanej metodzie – FastText



Rysunek 3.1: Architektury CBOW oraz Skip-gram [28].

3.1.2. FastText

Kolejnym podejściem do zanurzeń słów jest Fastext [35], inspirowany Skip-gramowym modelem Word2Vec (wspiera również podejście CBOW). Poprzednia metoda traktuje słowa jako pojedyncze tokeny, nie potrafi łatwo powiązać tej samej formy gramatycznej. Przykładem jest taka sama końcówka różnych czasowników z tą samą formą, np. „one kupują" oraz „one biegają". W tym modelu, każde słowo jest dzielone na zbiór n-gramów na literach. Dodatkowo, oryginalnie zastosowano znaki $<$ oraz $>$ jako oznaczenia początku i końca słowa. Przykładowo, słowo „nie" może wystąpić jako początek słowa „niebieski", koniec słowa „podanie", środek w „konieczność", czy w końcu samodzielnie jako „nie". Powyższe oznaczenia pozwalają rozróżnić te sytuacje.

Zanurzenia powstają dla każdego z n-gramów, reprezentacja słów to suma wektorów n-gramów, z których zbudowane jest słowo. Model ten pozwala dzielić ciągi liter między wieloma słowami, co pozwala bardziej wiarygodnie budować reprezentację wektorową również dla rzadkich słów. Kluczowe jest to dla niewielkich zbiorów danych o dużej liczbie unikalnych słów. Warto zwrócić uwagę na możliwą liczbę różnych n-gramów na literach. Ta liczna może wynosić wiele milionów, dlatego autorzy oryginalnej pracy zastosowali ograniczenie liczby możliwych zanurzeń, powstanie ich tylko B, gdzie B to wybrana stała. W pracy prezentującej pomysł FastText użyto $B = 2 \cdot 10^6$. Ograniczenie to jest zaimplementowane poprzez haszowanie. Każdy n-gram trafia do jednego z B koszyków poprzez zastosowanie pewnej funkcji haszującej. Można tu zauważyć podobieństwo z architekturą PRADO, gdzie również

używamy funkcji haszującej łączącej w jeden koszyk niezwiązane ze sobą elementy.

Dla dwóch powyższych podejść do zanurzeń słów istnieje bardzo wygodna implementacja powyższych zanurzeń za pomocą biblioteki *gensim*, która została wykorzystana w pracy.

3.1.3. GloVe

Innym rozwiązaniem jest GloVe (Global Vectors) [36]. W porównaniu do Word2Vec, w tej metodzie chcielibyśmy uzyskać nie tylko lokalne informacje o wektorze słowa, ale także takie o szerszym zasięgu. W badaniach można rozróżnić dwa podejścia: związane z faktoryzacją macierzy takie jak *latent semantic analysis* (LSA) oraz bazujące na lokalnym otoczeniu słowa, takie jak skip-gram.

Głównym celem jest połączenie tych dwóch światów poprzez zastosowanie modelu dynamicznej regresji logistycznej. Funkcja celu jest następująca:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log(X_{ij}))^2 \tag{3.5}$$

funkcja f to pewna funkcja skalująca zależna od wartości w macierzy X, X to macierz współwystąpień słów w kontekście innych słów, w_i to wektory słów, b_i to termy typu *bias*. Funkcja przypomina regresje liniową.

Intuicje za użyciem macierzy współwystąpień jest następująca: mając dane słowa i=*ice* oraz j =*steam*, chcemy zbadać stosunek prawdopodobieństw współwystąpień mając do dyspozycji słowo k =*solid*. Z uwagi na to, że słowo *ice* spotkamy częściej ze słowem *solid*, oczekujemy, że $X_{ik} > X_{jk}$. Przeciwna sytuacja wystąpi ze słowem l =*gas*, $X_{jl} > X_{il}$.

Probability and Ratio	k = *solid*	k = *gas*	k = *water*	k = *fashion*
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Rysunek 3.2: Prawdopodobieństwa współwystąpień słów *ice* i *steam* w kontekście innych słów. [36].

3.2. Locality-sensitive hashing

Inne podejście do wyznaczania wektorowych zanurzeń słów wywodzi się z zadania wyszukiwania najbliższych sąsiadów. W przypadku NLP, możemy mówić o

szukaniu podobnych dokumentów czy też plików audio z ludzką mową. Jednym z takim algorytmów jest Locality-sensitive hashing (LSH). Nawiązuje on do rodziny funkcji haszujących, które mają podzielić dane na koszyki (buckets) w następujący sposób: punkty znajdujące się blisko siebie znajdą się w takim samym koszyku z wysokim prawdopodobieństwem, natomiast punkty o dużej różnicy odległości najprawdopodobniej znajdą się w innych koszykach.

O LSH najczęściej mówi się kontekście podobieństwa dokumentów [37], ale my skupimy się się na odmianie LSH, która działa w przestrzeni n-wymiarowej wektorów o współrzędnych rzeczywistych pozwalającej na znalezienie podobnych wektorów $\mathbb{R}^n$.

Rozważmy pewną losową hiperpłaszczyznę przechodzącą przez punkt $\mathbf{0}$. Hiperpłaszczyznę możemy reprezentować poprzez wszystkie wektory prostopadłe do jej wektora normalnego zaczepione w punkcie $\mathbf{0}$. Wtedy LSH będzie działać następująco: dwa wektory uznany za podobne, jeśli leżą po tej samej stronie hiperpłaszczyzny, co możemy sprawdzić poprzez znak iloczynu skalarnego. W przypadku znaku dodatniego zwrócimy 1, 0 w przeciwnym wypadku. Procedurę powtarzamy d razy, gdzie d to szukana liczba wymiarów zanuzeń.

3.2.1. Modyfikacje LSH - wariant 1

Możemy wprowadzić pewną modyfikację i uprościć proces - zamiast losować wektory tworzące hiperpłaszczyzny, możemy losowo wybrać jeden z istniejących wektorów zanurzeń oraz względem niego liczyć podobieństwo cosinusowe pozostałych wektorów. Wszystkie wektory o podobieństwie większym od pewnego progu t otrzymają 1, pozostałe 0, Powtarzając losowanie n razy, możemy stworzyć wektory składające się z zer i jedynek reprezentujące zanurzenia słów. Można również wybrać inne wektory, względem których będziemy liczyć podobieństwo. Przykładem są np. różnice istniejących dwóch zanurzeń.

3.2.2. Modyfikacje LSH - wariant 2

Oryginalnie w LSH rozważa się wektory zero-jedynkowe, ale nic nie stoi na przeszkodzie. aby rozszerzyć ten zbiór wartości. Proponujemy użycie wektorów zbudowanych na podstawie zanurzeń oraz procedury LSH składających się z trzech wartości $\{-1, 0, 1\}$. W tym celu będą potrzebne dwa progi wartości. Celem jest podzielenie zbioru wektorów na bliskie wektorowi rozpatrywanego w danym koszyku, takie które są związane z nim w niewielkim stopniu oraz na wektory niepodobne. Inspiracją jest procedura budowy projekcji w PRADO oraz praca o sieciach neuronowych z potrójnym wagami (ternary weight networks) [30].

Rozdział 4.

Dane i ich przetworzenie

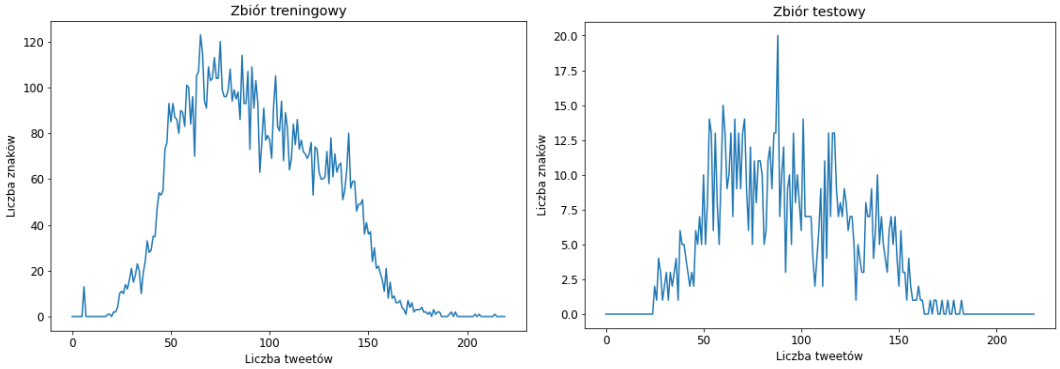
4.1. Zbiór danych do zadania

Autorzy zadania 6 z konkursu PolEval 2019 przygotowali zbiór danych, który jest głównym źródłem danych w pracy. Dane zostały już wcześniej podzielone na zbiór treningowy i testowy przez autorów, co miało pozwolić na późniejszą obiektywną ocenę wyników w porównaniu do innych uczestników konkursu. Oczywiście dane testowe nie były publikowane przed rozstrzygnięciem konkursu. Komentarze pochodzą z 19 najbardziej popularnych oficjalnych kont na Polskim Twitterze w roku 2017. Są to zarówno posty, jak i komentarze do nich. Pod uwagę zostały wzięte liczby obserwujących, aktywność, liczba komentarzy oraz wspomnień. Są to następujące konta:

@tvn24, @MTVPolska, @lewy_official, @sikorskiradek, @Pontifex_pl, @donaldtusk, @BoniekZibi, @NewsweekPolska, @PR24_pl, @tvp_info, @rzeczpospolita, @AndrzejDuda, @lis_tomasz, @K_Stanowski, @R_A_Ziemkiewicz, @pisorgpl, @Platforma_org, @RadioMaryja, @RyszardPetru.

Surowe dane zostały poddane wstępnemu przetworzeniu. Usunięto tweety z dużą liczbą skrótów, niepełne zdania, krótsze niż 5 słów, a następnie zanimizowano odniesienia do innych kont. Ciekawe podejście zastosowano do tweetów z adresami URL, które również zostały pominięte. Wynika to z tego, że adresy URL często są długie, a tweety w zdecydowanej większości nie są dłuższe niż 140 znaków, a ich maksymalna możliwa długość to 280 znaków [38]. Usunięte zostały również wypowiedzi zawierające tylko wspomnienia (@) oraz hasztagi (#) [17]. Dla dyspozycji uczestników zostało oddanych 10041 tweetów w zbiorze treningowym oraz 1000 w zbiorze testowym. Zostały one dostarczone w czterech plikach tekstowych, po dwa na zbiór, jeden z tekstami w kolejnych liniach, a drugi z klasami (0 - normal, 1 - hate).

Warto wspomnieć o procesie adnotacji. W celu zachowanie jak największej obiektywności w ocenie, wprowadzono kilka wskazówek, co powinno być sklasyfi-



Rysunek 4.1: Liczba znaków tweetów w zbiorach danych

kowane jako *hate*. Poniżej kilka z nich.

1. wyłudzanie, ujawnienie lub groźba ujawnienia prywatnych informacji
2. atak personalny ("Powieś się, gnoju![1])
3. groźby ("znajdę cię i zajebie") szantaże ("powiem wszystkim gdzie mieszkasz, jeśli mi nie zapłacisz")
4. szyderstwa/wyśmiewanie ("Patrzcie na tego grubasa", "ty pryszczata mordo")

4.2. Inne dostępne zbiory danych

W porównaniu do języka angielskiego, w języku polskim istnieje jedynie kilka zbiorów danych z przygotowanymi adnotacjami z dziedziny analizy wydźwięku. Udało się się znaleźć jedynie kilka zbiorów wypowiedzi mowy nienawiści w języku polskim poza zbiorem z zadania 6.1 [39, 40]. Istnieje jeszcze kilka zbiorów z recenzjami produktów, ale nie są już one przydatne w opisywanym zagadnieniu [41, 42]

4.3. Analiza komentarzy w zbiorze danych

W surowych plikach znajduje się 11 041 tweetów, 10 041 w zbiorze treningowym oraz 1000 w testowym. Jednak kilka z nich trzeba wykluczyć z uwagi na duplikację (retweety). Zmniejsza do liczbę wypowiedzi odpowiednio do 9387 i 945 w zbiorze treningowym i testowym.

Skupmy się na długości dostępnych wypowiedzi. Na rysunku 4.1 wyraźnie widać, że mimo zwiększenia długości tweetów do 280, to wciąż część wypowiedzi mieści

[1]Z uwagi na temat pracy, wulgaryzmy i obraźliwe określenia są nieuniknione, za co przepraszamy czytelnika

się w poprzednim limicie 140 znaków. Jedynie odpowiednio 9.3% i 9.4% wypowiedzi przekracza długość 140 znaków w zbiorze treningowym i testowym. Najdłuższa wypowiedz ma 215 znaków.

Ważną dla zadania i budowania modeli kwestią jest balans klas, wykorzystywany zbiór jest mocno niezbalansowany. W zbiorze treningowym mamy 8% wypowiedzi typu *hate*, natomiast w testowym 12.8%. Pozostałe komentarze i posty to typ *normal*.

W tabeli 4.1 widzimy kilka przykładów surowych wypowiedzi z dwóch klas. *@aa* reprezentuje skrót od *@anonymized_account*, oznaczającą dowolną wzmiankę. Już na tych tekstach możemy zauważyć kilka interesujących faktów:

- przykłady 1 i 2 mimo zawierania słów wulgarnych nie są oznaczone jako *hate*
- interpunkcja nie jest zawsze przestrzegana, pojawiają się literówki, zob. teksty 4, 6, 7
- oznaczenie wypowiedzi 4 i 5 jest subiektywne, z nieco innymi kryteriami oceny mogłyby mieć inną klasę
- pojawiają się wyrazy nacechowane negatywnie rzadziej stosowane na co dzień, zob. komentarz 7

Wszystkie te uwagi mają zastosowanie do całego zbioru treningowego.

Tablica 4.1: Przykładowe komentarze - zbiór treningowy

Id	Tekst	Klasa
1	@aa Wojna potrzebna, a narodowiec "farbowany lis"bedzie pierwszy spierdalał	normal
2	@aa @aa Za dużo czytam GP,Sakiewicza i Wolskiego, odjebalo mi sorry	normal
3	@aa Nie każdy pit jest taki prosty	normal
4	@aa Wstydze się że jest Pan w parlamencie	normal
5	@aa Kolejny filar inteligencji z PiSu	hate
6	@aa Straszy redekcje karami gałgan jeden	hate
7	#BezRetuszu Postkomunistyczne układy, sitwy,kasty wszystko to na śmietnik historii !! Natychmiast !!	hate
8	@aa A kto kutasie sprzedał Stocznię Ukraińcom?	hate

4.4. Wstępne przetwarzanie tekstów

Z uwagi na surowość danych potrzebne było odpowiednie wstępne przetworzenie danych przed uruchomieniem modelu. Pierwszym krokiem było podzielenie

tekstu na pojedyncze tokeny, za separatory zostały uznane spacje i jej wielokrotności i/lub znaki interpunkcyjne. Warto wspomnieć o dwóch odmiennych sposobach wykonania tej operacji. Możemy najpierw podzielić tekst na tokeny, a później usunąć interpunkcję lub zrobić dwa te kroki jednocześnie. Różnica w wyjściowym tekście jest następująca: w pierwszym podejściu w przypadku znaku interpunkcyjnego w środku wyrazu znak zostanie usunięty i powstanie jedno słowo, natomiast w drugim podejście znak będzie uznany za separator i powstaną dwa osobne tokeny. Wstępne eksperymenty pokazały gorsze wyniki przy zachowaniu interpunkcji, więc została ona usunięta. Wielkie litery zostały zastąpione małymi odpowiednikami.

Następne kroki obejmowały analizę form gramatycznych. Z uwagi na wysoki stosunek liczby unikalnych słów w porównaniu do całkowitej liczby słów, podjęto próbę zmniejszenia tej liczby. W tym celu użyty został *Analizator i generator fleksyjny Morfeusz 2* [43]. Dla każdego słowa w bazie danych możemy otrzymać dokładną formę gramatyczną, wraz z lematem słowa. Lemat słowa to jego forma podstawowa, np. mianownik rzeczownika czy bezokolicznik czasownika. We wszystkich możliwych przypadkach słowa odmienione zostały zastąpione przez lemat. Pozwoliło to zmniejszyć liczbę unikalnych słów z ponad 31 tys. do niecałych 14 tys. Warto wspomnieć o tym, że niektóre słowa mogą mieć kilka lematów, w takim wypadku użyty został pierwszy lemat wypisany przez *Morfeusza*. Tak przykładowo może wyglądać zlematyzowany tekst, z usuniętą interpunkcją:

1. Może gustował w starszych paniach ;-) → może gustować w starszy pani
2. Joanno! Po raz pierwszy się z panią zgadzam. → joanna po raz pierwszy się z pani zgadzać

W przypadku emotikon zastosowano kilka różnych podejść. Pierwsze z nich je ignorowało, kolejne próbowały zachować je w tekście. Druga metoda korzystała z gotowych określeń emotikon pod jakimi są zdefiniowane w Unicode. Niestety są one w języku angielskim, natomiast wszystkie wypowiedzi w języku polskim. Dlatego zaproponowaliśmy również kolejne podejście: podmienienie emotikon na polskie słowa odpowiadających emocjom. Słownik ten został stworzony ręcznie.

W celu zminimalizowania wpływu braku balansu klas, wypróbowaliśmy również multiplikację komentarzy typu *hate* w zbiorze treningowym. Zastosowaliśmy również kilka metod radzenia sobie ze słowami występującymi w zbiorze jedynie 1-2 razy. W podstawowej wersji nie były one zmieniane, w kolejnej wszystkie rzadkie słowa zostały zamienione przez jeden specjalny token, w ostatniej zaś skorzystaliśmy z *Morfeusza* i proponowanej formy gramatycznej zamiast słowa.

Powyższe metody zostały też zastosowane w stosunku korpusu udostępnionego w ramach zadania 3 z PolEval 2018 [44], który służył jako rozszerzenie bazy tekstów dla zanurzeń słów.

Rozdział 5.

Przedstawienie eksperymentów

W tej części porównamy otrzymane wyniki dla różnych modeli skupiając się na zanurzeniach słów. Głównym zbiorem testowym jest osobny plik przeznaczony tylko do ewaluacji udostępniony przez organizatorów konkursu PolEval 2019. Na tym samym zbiorze były oceniane metody uczestników konkursu. Metody będziemy umieszczać w rankingu, aby sprawdzić, które miejsce moglibyśmy zająć w zawodach, gdybyśmy wysłali konkretny model. Z uwagi na chęć porównania się do uczestników, zależy nam na wynikach na zbiorze testowym, co wzbudza ryzyko przeuczenia i zbytniego dostosowania się do danych. Z uwagi na ograniczenia pamięci stosujemy mniejsze niż częściej stosowane zanurzenia, proponujemy długość 128 i 256. Z uwagi na dużą dysproporcję między klasami w zbiorze testowym (jedynie 12.8% *hate*) główna miara oceny jakości modeli to F1-score. Klasą celu był *hate*. W opisie pojawiają się wyrażenia TP – TruePositive (poprawnie sklasyfikowany hate), FP – FalsePositive (niepoprawnie przypisane zwykłe wypowiedzi), TN – True Negative (dobrze sklasyfikowane zwykłę teksty), FN - False Negative (niewykryty hate).

Obliczenie zostały wykonywane na lokalnej maszynie z procesorem Intel Core i7-8565U CPU @ 1.8GHz z 24 GB pamięci RAM oraz z wykorzystaniem usługi Google Colab z pamięcią RAM w ilości 12 GB.

5.1. Otrzymane wyniki

5.1.1. Naiwny klasyfikator bayesowski

Testy zaczynamy od jednego z najprostszych modeli: Naiwny klasyfikator bayesowski, który korzysta z twierdzenia Bayesa oraz założenia niezależności pomiędzy kolejnymi słowami Jedyną cechą jest liczba wystąpień słów w korpusie. Jest to główna cecha dla pozostałych klasycznych metod - drzew decyzyjnych, SVMs oraz regresji logistycznej. Zaczynamy od zastosowania wstępnego przetwarzania opisanego w Rozdziale 4 tj. sprowadzenia słów do form bazowych, usunięcia interpunkcji,

wielkich liter, wzmianek, nie uwzględnianie emotikon. Udało mu się poprawnie sklasyfikować 6 wypowiedzi typu hate, $F1 = 8.9$.

1. @aa Półgłówek Wieliński, wymyślił sobie półautorytaryzm
2. @aa @aa @aaa I wojna Gadowskiego z psychiatrą.
3. @aa @aa @aa Zreszta ty chuja zobaczysz, kutasa ziobry najwyzej
4. @aa @aa Polska jest jedną? Dla protestujących w sejmie też? Ty kłamliwa gnido
5. @aa #PiSkomuna zdycha w lękach, Nasze słowa - ich udręka
6. @aa @aa @aa Kwalifikacje pogodynki umożliwiają w PiSie wszystko

W wyżej wymienionych tekstach możemy zauważyć kilka cech: odniesienia do polityki, negatywnie nacechowana słowa w tym wulgaryzmy, nazwiska. Są one też krótkie, większość słów posiada wymienione cechy.

Z uwagi na brak balansu klas, przy tworzeniu korpusów dla klas możemy zwiększyć wagę słów trafiających do zbioru *hate*, np. do 2. W ten sposób udało się wykryć już 13 wypowiedzi nienawistnych oraz osiągnąć $F1 = 17.6$. Można też dodać pewną modyfikację do algorytmu – jeśli słowo często występuje w korpusie *hate*, ale nie występuje w korpusie *normal*, to uznajemy wypowiedź za negatywną. Stała wybrana eksperymentalnie to 5. Tutaj udało się poprawić wynik do $F1 = 20.0$, 15 poprawnie sklasyfikowanych przykładów mowy nienawiści. Przykłady rozszerzają spostrzeżenia do poprzednich, wciąż jednak jesteśmy na ostatnim miejscu w konkursie.

Przypomnijmy teraz możliwe warianty dzielenia tekstu i nazwijmy je dla czytelności. Metoda dzieląca tekst na znakach interpunkcyjnych i spacjach to *Przetwarzanie 1*, natomiast ta tylko na spacjach to *Przetwarzanie 2*. Powyższe wyniki zostały uzyskane przy Przetwarzaniu 1 i jest to domyślna metoda z uwagi na lepsze wyniki na wielu modelach na zbiorze treningowym.

1. @aa Konserwator kuwety, gegacz i przechył mózgowy, za pieniądze robił loda w tokfm
2. @aa @aa A potem pójdziesz pod prokuratora za ukradzione kilometrówki
3. @aa @aa @aa @aa Ty i Kukiz15 już skończył. Zostaliście zjedzeni, przetrawieni i wysrani
4. @aa @aa @aa @aa Jacy delikatnie, ojej a kobiety opluwać, ciągać szmaciarze potraficie

W Tabeli 5.1 widzimy porównanie kolejnych podejść.

Tablica 5.1: Porównanie podejść Naiwnego klasyfikatora bayesowskiego. Poszczególne kolumny to: Metoda, Pr – Precision, Rc – Recall, F1 – F1-score, Acc. - Accuracy (dokładność), Pos - Position (pozycja w konkursie, gdyby ta metoda została wysłana). Oznaczenia pojawiają się także w kolejnych tabelach

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
z przetw.	42.8	5.0	8.9	87.0	**15**
z przetw.+2×hate	48.1	10.7	17.6	87.1	**15**
z przetw.+2×hate+częsty hate	**51.7**	**12.4**	**20.0**	**87.4**	**15**

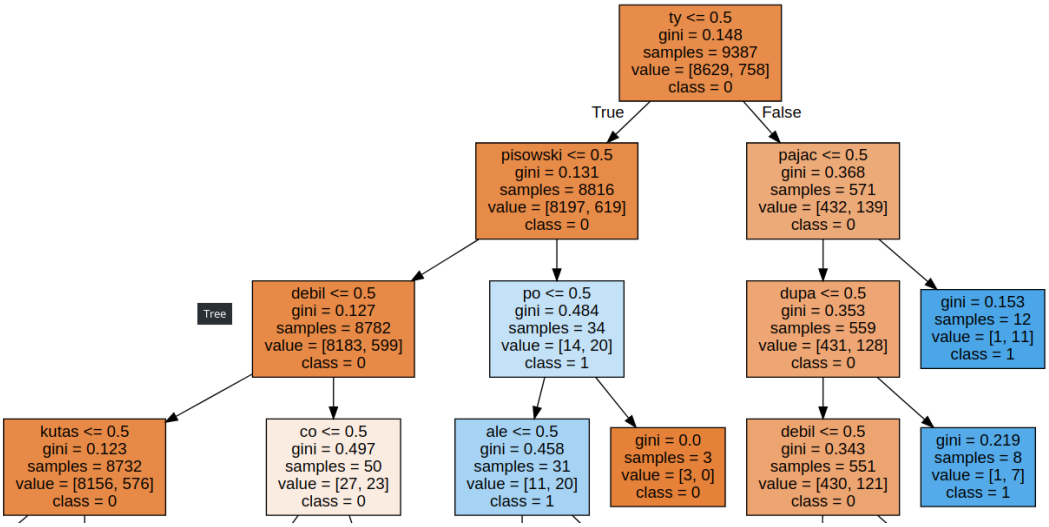
5.1.2. Drzewa decyzyjne

Kolejnym testowanym modelem z dziedziny algorytmów klasycznych są drzewa decyzyjne. Pierwsze podejścia obejmowały tylko jedno drzewo, gdzie dane zostały już wstępnie przetworzone. Do wektoryzacji słów została wykorzystana rzadka macierz wystąpień. Zacznijmy od małej głębokości równej 6. Wyniki są gorsze niż przy Naive-Bayesie, jedynie $F1 = 10.9$. Warto zwrócić na to, jakie słowa są w korzeniu tego drzewa 5.1. Słowa powszechnie uznawane za nieprzyjemne nie stanowią zaskoczenia, ale pierwsze słowo w korzeniu może być zadziwiające – „ty". Sugeruje to wypowiedzi skierowane to konkretnej osoby, aż 18% tekstów typu *hate* w zbiorze treningowym zawiera co najmniej jedno wystąpienie tego słowa.

Naturalnie możemy zwiększyć głębokość drzew. Na zbiorze testowym już przy głębokości 10 mamy $F1 = 19.1$, przy 15 mamy $F1 = 25.2$, przy 20 $F1 = 27.6$, w końcu z głębokością 25 $F1 = 29.7$. Mimo, że wyniki są coraz lepsze, to tak duże głębokości stwarzają ryzyko przeuczenia, szczególnie biorąc pod uwagę, że zdecydowana większość tweetów jest krótsza niż 30 słów. Lepsze wyniki zawdzięczamy głównie wyższemu Recall, udało się rozpoznawać więcej nienawiści bez zwiększania wartości FalsePositive. Ponadto drzewa o głębokości większej niż 15 są na 13 miejscu w konkursowym rankingu. W przykładach widzimy, że nie wszystkie słowa muszę być skojarzone z *hate*, aby wypowiedź został poprawnie sklasyfikowana. Poniżej przykłady wykrytej mowy nienawiści:

1. @aa @aa @aa Najbardziej to on jest wolny od mózgu.
2. @aa @aa @aa Ty nie jesteś Zero. Ty jesteś poniżej zera bezwzględnego
3. @aa @aa @aa tak minister edukacji uczy dzieci. Kolejny pisowski kłamca, oszust i złodziej.
4. @aa Wybaczam ci że nie masz swastyki na ryju i na profilowym, mimo skarg

Z uwagi na obawy o przeuczenie, warto wypróbować *pruning*, czyli obcinanie takich gałęzi, które zawierają mniej wypowiedzi niż pewna określona stała, w naszym



Rysunek 5.1: Korzeń płytkiego drzewa decyzyjnego

przypadku 5. Niestety ta metoda nie polepszyła naszych wyników, były one takie same bądź nieco gorsze niż bez niej. Parametrem, który możemy zmienić jest metoda wektoryzacji. Spójrzmy na TF-IDF (Term Frequency Inverse Document Frequency.) W założenia ta metoda ma lepiej oddawać wagi słów, które są ważne dla klasyfikacji. Przy tej modyfikacji udało się poprawić wyniki dla drzew o głębokości 10 i 15, mamy teraz odpowiednio $F1 = 21.7$ i $F1 = 26.8$. Pozostałe wyniki pozostały na podobnym poziomie, wykryte wypowiedzi również bez zmian.

Z uwagi na dobre wyniki jednego drzewa i ryzyko przeuczenia, warto zwrócić uwagę na klasyfikatory składające się z kilku. Jedną z metod jest bagging, czyli uwzględnienie wyników kilku klasyfikatorów uczących się na podzbiorze danych treningowych. Ostateczny wynik jest wybierany przez głosowanie większościowe. Przy baggingu mamy do wybory kilku parametrów: głównymi są liczba używanych drzew oraz wielkość podzbioru danych w jednym klasyfikatorze. Dla wartości 0.6-0.8 całego zbioru wypowiedzi wyniki są zbliżone. Zgodnie z oczekiwaniami wyniki dla mniejszej liczby drzew są niestabilne, dlatego ich liczba została zwiększona. Przykładowa dla proporcji 0.6, przy 10 drzewach $F1$ jest między 21.4, a 25.5 a przy 100 stabilizuje się w okolicach $F1 = 24.3$.

Z uwagi na brak balansu klas, pomysłem na modyfikację jest zastosowanie wielu drzew decyzyjnych losując pewną liczbę komentarzy *hate* i taką samą liczbę wypowiedzi zwykłych. Następnie predykcja zostanie ponownie wyznaczona poprzez głosowanie większościowe. Model z 10 drzewami, gdzie każde z nich zawiera w treningu 70% dostępnego *hate* i tyle samo wypowiedzi zwykłych, zdołał poprawić poprzednie podejścia do poziomu $F1 = 35.0$. TF-IDF nie wpływał na wyniki. Pruning w przypadku wielu drzew nie jest sensownym podejściem, gdyż sama liczba drzew jest czynnikiem regularyzacji. Eksperymentalnie wystarcza 200-300 drzew do ustabilizo-

wania wyników. Najlepszy wynik to $F1 = 40.0$, lepszy rezultat uzyskujemy poprzez poprawienie Recall, jednocześnie tracimy Precision. Kilka przykładów pokazuje problemy z literówkami (zdanie 1), czułość na wulgarne słowa (zdanie 3) czy kwestia słowa "ty". Poniżej kilka przykładów wyników klasyfikacji na zbiorze testowym.

1. Jak widzę byłego premiera Marcinkiewicza to zawsze myślę: qrwa!jak ktoś taki mógł być premierem. Taki skończony błazen. – FN

2. @aa A kto traktuje poważnie ten szmatławiec?? – FN

3. Jak się kurwa zapisuje na wf – FP

4. @aa Ty najpierw przeczytaj te swoje książki z fototapety a potem pogadamy. – FP

Tablica 5.2: Porównanie wariantów drzew decyzyjnych

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
1 drzewo gł.6	**87.5**	5.8	10.9	87.8	15
1 drzewo gł.15	81.8	14.9	25.2	88.7	13
1 drzewo gł.25	81.5	18.2	29.7	**89.0**	13
1 drzewo gł.25 + pruning	78.6	18.2	29.5	88.9	13
1 drzewo gł.15 + TF-IDF	71.4	16.5	26.8	88.5	13
bagging, 50 drzew	79.2	15.7	26.2	88.7	13
10 drzew zbalansowanych, gł.15	38.2	32.2	35.0	84.7	**12**
300 drzew zbalansowanych, gł.25	45.7	**35.5**	**40.0**	86.3	**12**

5.1.3. Support-Vector Machines

Inną użytą metodą były Support-Vector Machines, SVMs (spotyka się czasem tłumaczenia *maszyny wektorów nośnych/wspierających*). Testy zaczynamy od jądra liniowego, daje ono całkiem dobre rezultaty, $F1 = 37.1$. Dodanie emotikon do danych umożliwia podniesienie wyniku do $F1 = 39.3$. Ciekawym parametrem, który zapewnia biblioteka sklearn jest parametr `class_weight='balanced'`. Umożliwia on dopasowanie wag klas odwrotnie proporcjonalnie do częstotliwości klas w zbiorze danych, w naszym przypadku klasa *hate* będzie miałą wyższą wagę. Ta prosta modyfikacja poprawiła wynik do $F1 = 42.9$. Próbowaliśmy również zmienić kernel na *rbf* (Radial Basis Function) oraz na klasycznego sigmoida, mamy odpowiednio do $F1 = 0.64$ (bardzo słaby rezultat) oraz $F1 = 15.5$.

Tablica 5.3: Porównanie wariantów SVM. Skrót emot. oznacza umieszczenie w danych emotikon.

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
jądro liniowe	67.4	25.6	37.1	**88.9**	12
jądro lin. + emot.	70.2	27.3	39.3	87.0	11
jądro lin.+emot.+balanced weights	58.6	**33.9**	**42.9**	88.5	**10**
jądro rbf+emot.+balanced weights	**100**	3.3	6.4	87.6	15
jądro sigmoid.+emot.+balanced weights	52.4	9.1	15.5	87.3	15

5.1.4. Regresja logistyczna

Interesujące wyniki udało się uzyskać przy klasycznej regresji logistycznej. Standardowa implementacja dała $F1 = 27.6$, ale zastosowanie podobnie modyfikacji wag dla różnych klas jak w SVM pozwolił uzyskać aż $F1 = 49.3$ na zbiorze testowym. Dodanie emotikon do zbioru delikatnie podniosło ten wynik do $F1 = 50.7$.

Tablica 5.4: Porównanie wariantów regresji logistycznej

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
standard	**83.3**	16.5	27.6	**88.9**	13
balanced weights	52.8	**46.3**	49.3	87.8	7
balanced weights + emotikony	56.0	**46.3**	**50.7**	88.5	**6**

5.1.5. Oryginalne PRADO

Przejdźmy teraz do wyników osiągniętych z architekturą PRADO. Sama implementacja udostępniona w serwisie GitHub [45] zawiera wiele różnych parametrów do wyboru, nasze badania rozpoczniemy od przykładowych notatników udostępnionych przez twórców pakietu. Już z podstawowym przetwarzaniem tekstów osiągamy $F1 = 50.8$, dałoby by to 6 miejsce w konkursie. Sprawdziliśmy tutaj też działanie przy Przetwarzaniu 2 – tylko $F1 = 45.4$.

Spróbujmy dodać emotikony w sposób wspomniany wcześniej, poprzez zastąpienie ich polskimi słowami opisującymi emocje. Pozwala to osiągnąć $F1 = 52.1$. Wypowiedzi w zbiorze danych nie są długie, maksymalna długość po przetwarzaniu to 32 tokeny, możemy zmniejszyć oczekiwaną długość zdań w parametrach modelu. Co ciekawe, wydajność spada po takiej zmianie, odpowiednio $F1 = 49.7$ przy długości 64 i do $F1 = 44.6$ przy 32 słowach. Nie pomaga również zwiększenie liczby cech w projekcji z 512 do 1024 ($F1 = 50.2$), ani dyskretyzacja zbioru wartości dla sieci ($F1 = 44.8$).

Zwróćmy uwagę na ogólną dokładność sieci, odbiega ona od wcześniej opisanych drzew. Wyższe wyniki wynikają głównie z większej wartości Recall, znacznie spadła

liczba niepoprawnie sklasyfikowanych wypowiedzi typu *hate*. PRADO z dodanymi emotikonami do danych wykryło 26 wypowiedzi typu *hate* więcej, niż zbiór zbalansowanych drzew, nie wykrywając tylko jednej wykrytej przez drzewa. Przykłady są następujące:

1. @aa Dalej biedny, hańba polski, Gowin nie ma na skromne święta
2. @aa towarzystwo będzie najlepsze jak znikniesz z horyzontu Panie Lis
3. @aa I tak dobrnęliście na dno szamba.
4. Moja przyjaciółka ma dzisisj poprawkę z pozytywizmu u najgorszego idioty na uczelni, mam nadzieje źe zda

Nie zawierają one oczywistych wulgaryzmów, ich negatywność jest zdecydowanie bardziej subtelna.

Tablica 5.5: Porównanie podejść PRADO bez modyfikacji. Skrót sen. to maksymalną oczekiwana długość tekstu w sieci

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
baza	47.5	54.5	50.8	86.4	10
baza + emot.	49.2	**55.4**	52.1	87.0	**4**
emot. + sen. 32	**65.1**	33.9	44.6	**89.2**	10
emot. + 1024 cechy	51.8	48.8	50.2	87.6	6
emot. + dyskretny	56.2	37.1	44.8	88.2	10

5.1.6. Zanurzenia Word2Vec

W celu utworzenia wartościowych wektorów zanurzeń, musimy mieć większy zbiór treningowy, który zawiera niecałe 10 tysięcy wypowiedzi. W tym celu wykorzystujemy opisany w Rozdziale 4 korpus PolEval 2018, pierwsze 5 mln wierszy [44]. Z uwagi na ograniczenia sprzętowe nie udało się pracować na cały korpusie (24 mln wierszy). Pozostałe metody zanurzeń również korzystały z tego korpusu.

Pierwsze podejście korzysta z bazowych parametrów proponowanych przez twórców biblioteki *gensim*. Korzystamy z modelu Word2Vec, długość kontekstu to 5 słów, typ to CBOW, długość wektorów zanurzeń to 100. Wynik to $F1 = 38.6$, nieco gorszy niż przy drzewach zbalansowanych. Naturalnym krokiem jest zwiększenie długości zanurzeń. Już przy długości zanurzeń równej 128 mamy $F1 = 42.5$, co poprawia wyniki najlepszych drzew.

Warto rozważyć wypróbowanie architektury skip-gram, zgodnie z literaturą ma ona lepiej działać w przypadku rzadkich słów, co jest przypadkiem w naszym zbiorze danych. Przy niezmienionych innych parametrach znacząco poprawiliśmy poprzedni

wynik uzyskując $F1 = 55.0$, co jest pierwszym sposobem poprawiającym PRADO z projekcją. Z tym modelem uzyskalibyśmy 3 miejsce w konkursie.

Parametrem, który możemy dostosować w modelu PRADO jest maksymalna oczekiwana długość wypowiedzi. Żadna z wypowiedzi nie przekracza 64 tokenów, możemy ją ustawić na taką wartość. Poprawiamy wtedy wyniki już do $F1 = 59.8$, co uczyniłoby ten model zwycięzcą konkursu. Naturalnym rozszerzeniem wydaje się być użycie dłuższych wektorów zanurzeń. Co ciekawe, przy długości zanurzeń równej 256 wyniki pozostają bardzo podobne, $F1 = 59.2$.

Podobnie jak w kolejnej metodzie, tutaj też próbujemy uzyskać dodatkową informację z emotikon, poprzez zastąpienie ich polskimi słowami opisującymi emocje. Dodatkowo, wzbogacamy korpus przez zbiór treningowy. Niestety, w tym przypadku $F1 = 57.5$.

W stosunku do PRADO przy architekturze skip-gram oraz maksymalnej długości wypowiedzi równej 64 główną różnica w wyniku jest niższy wskaźnik FP. PRADO aż 69 wypowiedzi z tagiem normalnym wskazała jako *hate*. Ponadto, pomimo niewielkiej różnicy w wynikach, klasyfikatory działają zupełnie inaczej. Przykładowo, z 69 wspomnianych wypowiedzi FP dla PRADO aż 50 z nich zostało poprawnie sklasyfikowane w przypadku word2vec. Nieco lepiej jest w przypadku FN. W PRADO w tej kategorii jest w sumie 54 wypowiedzi, 27 nie pojawia się w FN w Word2Vec, w drugą stronę na 48 tekstów w FN w Word2Vec, 21 jest dobrze klasyfikowanych w PRADO.

Tablica 5.6: Porównanie wariantów PRADO+Word2Vec. Tren. oznacza dodanie zbioru treningowego do korpusu

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
dł.100	48.2	32.2	38.6	86.9	11
dł.128	47.0	38.8	42.5	86.6	10
dł.128 + skip-gram	58.3	52.1	55.0	89.1	3
dł.256 + skip-gram	**62.6**	51.2	56.4	89.8	3
dł.128 + skip-gram + max.sen.len 64	59.4	**60.3**	**59.8**	89.6	**1**
dł.128+skip-gram + sen.64 + tren. + emot.	56.3	58.7	57.5	**88.9**	3
dł.256 + skip-gram + sen.64	59.7	58.7	59.2	89.6	**1**

5.1.7. Zanurzenia FastText

Wiemy już, że architektura skip-gram radzi sobie znacznie lepiej niż CBOW, więc przy zanurzeniach FastText też z niej korzystamy. Pierwsze wyniki zostały uzyskane dla kontekstu długości 5, długość wektorów zanurzeń jest równa 100, osiągnięto F1 na poziomie 54.1. Podobnie jak przy Word2Vec, wydłużmy wektory zanurzeń do 128 liczb. Tym razem $F1 = 54.8$. Niestety ograniczone zasoby nie pozwoliły bardziej zwiększyć długości wektorów. Podobnie jak wcześniej dodaliśmy również ogranicze-

nie na oczekiwaną długość wypowiedzi w PRADO - przy 64 tokenach $F1 = 56.2$, co jest podobnym wynikiem do Word2Vec.

Kolejnym pomysłem na możliwość poprawienia wyników jest wzbogacenie danych do trenowania zanurzeń. Zrobimy to poprzez dodanie zbioru treningowe do korpusu. Utrzymując poprzednie parametry, niestety mamy nieco gorsze wyniki - $F1 = 55.6$.

Podjęliśmy również próbę innego przedstawiania słów rzadkich o częstotliwości wystąpień jedynie 1-2 razy w zbiorze treningowym. Jedną z propozycji było zastąpienie wszystkich z nich za pomocą jednego specjalnego tokena, drugą zaś zastąpienie ich formami gramatycznymi pochodzącymi z Morfeusza. Obie metody nie poprawiły wyników.

Kolejna modyfikacja była motywowana chęcią utrzymania znaczenia emotikon w tekście. Początkowa propozycja zastąpienia ich identyfikatorami znajdującymi się w Unicode nie przyniosła oczekiwanej poprawy ze względu na to, że są one w języku angielskim i składają się z wielu słów. Dlatego własny pomysł zastąpienia emotikon odpowiednimi słowami reprezentującymi emocje okazał się lepszym podejściem. Udało się uzyskać $F1 = 60.0$ – zdecydowanie lepszy od poprzednich podejść z Word2Vec.

Przy porównaniu FastText z oryginalnym PRADO można wyciągnąć podobne wnioski jak przy porównaniu Word2Vec i PRADO - klasyfikatory działają bardzo różnie. Ciekawsze jest jednak porównanie Word2Vec i FastText. Na przykład dla TP przy skali ≈ 70 dobrze sklasyfikowanych wypowiedzi typu *hate*, jedynie 15 w TP FastTexta nie pojawia się w TP Word2Vec, w druga stronę wypowiedzi jest 16.

Tablica 5.7: Porównanie wariantów PRADO+FastText

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
dł.100	59.2	50.0	54.0	89.2	3
dł.128	57.9	53.0	54.8	89.0	3
dł128, sen.64	**66.3**	48.8	56.2	**90.3**	3
dł.128, sen.64, + treningowy	57.6	53.7	55.6	89.0	3
n=128, sent.len.=64 + tren.+ emot.	60.5	**59.5**	**60.0**	89.8	**1**

5.1.8. Zanurzenia GloVe

Sprawdźmy możliwości metody zanurzeń korzystającej z globalnych statystyk - GloVe. Zaczynamy od długości kontekstu równej 5 i długości wektorów zanurzeń równej 128. Korzystając z Przetwarzania 2, wyniki są spore gorsze niż przy poprzednich modelach, jedynie $F1 = 37.2$. Przy Przetwarzaniu 1 i niezmienionych pozostałych parametrach mamy $F1 = 42.0$, co również jest słabszym rezultatem.

Tablica 5.8: Porównanie wariantów PRADO+GloVe

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
dł. 128, przetwarzanie 2	45.8	31.4	37.1	**86.5**	12
dł. 128, przetwarzanie 1	**41.8**	**42.2**	**42.0**	85.1	**10**

5.1.9. Locality-sensitive hashing

Jako punkt wyjścia do LSH bierzemy jeden z lepszych zbiorów zanurzeń pod względem osiągniętych wyników - zanurzenia typu FastText o wektorach długości 128 z architekturą skip-gram. Z uwagi na charakter zadania klasyfikacji oraz chęci otrzymania wektorów z wartości ze zbioru $\{0, 1\}$ stosujemy opisane wcześniej modyfikacje LSH. Zaczynamy od Przetwarzania 2.

Z uwagi na dużo mniej wymagający pamięciowo proces, jesteśmy w stanie stworzyć kilka razy dłuższe wektory niż przy Word2Vec i FastText. Dodatkowo, przy tej samej długości, otrzymamy model może zostać efektywnie zakodowany na 32-krotnie mniejszej pamięci - zamiast kodować liczbę jako typ float na 32 bitach, możemy każdą wartość 0 i 1 zapisać jako jeden bit liczby całkowitej. W efekcie, wektor dla jednego słowa o długości 128 zajmie tylko 128 bitów (4 liczby całkowite).

Testy zaczniemy od LSH z modyfikacją nr 1. Losujemy zanurzenie słów ze zbioru treningowe, dla każdego innego słowa liczmy podobieństwo cosinusowe i na podstawie relacji z progiem umieszczamy 0 bądź 1. Próg został wybrany tak, aby około połowa słów miała na danej współrzędnej 1. Wydawałoby się, że poprzez zrezygnowanie z liczb typu float tracimy dużą cześć informacji, ale przy tym podejściu uzyskujemy $F1 = 55.2$, co umieszcza nas na 3 miejscu w konkursie. Podobnie jak przy poprzednich metodach, wydłużmy wektory zanurzeń do 1024 wymiarów. Tym razem mamy już $F1 = 58.6$, przy uwzględnieniu dalszych miejsc po przecinku byliśmy tuż za zwycięzcą - systemem ULMFiT.

Teraz spróbujmy podejścia, gdzie nie liczymy podobieństwa z jednym konkretnym wektorem, ale z pewną różnicą dwóch losowych wektorów zanurzeń. Intuicja jest następująca – $w[\text{król}] - w[\text{mężczyzna}] + w[\text{kobieta} \approx w[\text{królowa}]$. Przy takiej metodzie i długości zanurzeń równej 512 nie mamy lepszych wyników, ale wynik wciąż jest wysoki - $F1 = 51.3$.

Innym ciekawą próbą poprawy wyników jest zastosowanie modyfikacji nr 2 do LSH – zamiast dzielić hiperprzestrzeń wektorów zanurzeń na dwie części, podzielmy ją na 3. Punktem wyjścia będzie lepiej działająca metodą z losowym wektorem ze zbioru treningowe. Przy obliczaniu podobieństwa cosinusowe dynamicznie wyznaczamy takie progi dla wartości ze zbioru $\{-1, 0, 1\}$, aby podzielić zbiór słów na 3 równe części. Wyniki nie są zachwycające - $F1 = 52.2$ nie poprawia poprzednich podejść

Przeprowadźmy jeszcze testy z Przetwarzaniem 1, w poprzednich modelach da-

wała ona lepsze rezultaty. Tak też jest i w tym przypadku, dla LSH z wektorami o długości 512 i modyfikacji nr 1 $F1 = 59.3$, co poprawia wynik nie tylko Przetwarzania 2 przy tej samej długości zanurzeń, ale także tych dłuższych. Weźmy również zanurzenia FastText ze zbiorem treningowym w korpusie i zachowanymi emotikonami. Ta modyfikacja poprawiła poprzednie wyniki, osiągając $F1 = 60.4$, ponownie okazało się, że informacja o emotikonach jest przydatna.

W przypadku LSH szczególnie ciekawy jest model o $F1 = 59.3$. Nie jest to najlepszy wynik pod względem tej metryki, ale posiada on najwyższy Recall spośród wszystkich testowanych modeli - $Rc = 68.6$. Może to być szczególnie ważne, gdy chcemy wykryć jak najwięcej komentarzy *hate*, nawet kosztem stworzenia zbyt czułego algorytmu. Taki model może być przydatny np. przy tworzeniu usług skierowanych dla młodszych użytkowników, gdzie moderacja treści jest szczególnie ważna.

Popatrzmy na kilka przykładów, czego algorytm nie rozpoznał jako *hate*. Subiektywnie co najmniej część z nich nie musi być uznana jako *hate*, przykładowo wypowiedzi 5-6. W tekstach takich, jak 1-4 użytkownik języka polskiego rozpozna negatywne zabarwienie, ale dla modelu będzie to trudne zadanie. Komentarz 1 to porównanie inteligencji człowiek do zwierzęcia, zdanie 2 zawiera pejoratywne porównanie, trzecia wypowiedź nawiązuje to osób niepełnosprawnych, a czwarta zawiera eufemizm - wszystkie te 4 przykłady są bardzo subtelne i wymagają wiedzy o całym języku. Niestety, nasze modele mają problem z takimi zdaniami.

1. @aa @aa @aa @aa Jaki tam profesor!!!To kangur którego. Aborygeni zamienili w człowieka.
2. @aa @aa Niestety nie oglądałem. Ale jedna rzecz w tym poście jest nieprawdziwa. Żalek i wiedza to dwa wykluczające się byty
3. @aa Jezdzisz na wózku? Tramwaj uciał ci nogi?
4. @aa Jako obywatel kategorycznie domagam się abyś włożył sobie nos w końcowy odcinek układu pokarmowego
5. @aa nie kłóć się jak nie chcesz mieć później problemów! :) @aa @aa @aadisc
6. @aa @aa @aa Przecież to proste - wystarczy nie kraść

Tablica 5.9: Porównanie podejść modyfikacji LSH z PRADO

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
dł. 512, losowy wektor, Przetw.2	62.2	46.3	53.1	**90.0**	4
dł. 1024, losowy wektor, Przetw.2	54.2	63.6	58.6	88.5	2
dł. 512, różnica wektorów, Przetw.2	**67.6**	41.3	51.3	**90.0**	6
dł. 512, 3 wartości, Przetw.2	48.3	57.0	52.2	86.7	4
dł. 512, losowy wektor, Przetw.1	52.2	**68.6**	59.3	88.0	**1**
dł. 512, Przetw.1+tren.+emot.	57.5	63.6	**60.4**	89.3	**1**

Spójrzmy na kilka przykładów FalsePositive, tzn. na komentarze, które zostały niepoprawnie oznaczone jak *hate*. Pierwsza wypowiedź jest wypełniona wulgaryzmami, dlatego nie dziwi pomyłka modelu. Podobnie dla wypowiedzi nr 5. Subiektywnie możemy stwierdzić, że wypowiedzi 3 i 4 mogłyby być uznane za mowę nienawiści przez czytelnika, dlatego możemy tutaj myśleć o błędzie w adnotacji klas. Wypowiedź 2 zawiera znowu odwołanie do partii politycznej, która często pojawia się negatywnym kontekście.

1. No teraz kurwa jak muszę wyjść to padać zaczyna no ja pierdole do dupy z takim życiem

2. @aa @aa Dołącz do kolegów w PiSie

3. @aa Ten to już zupełnie odwiesił mózg na kołek, chory mózg

4. @aa Bo ty tam pracujesz. Oszołomie

5. Jak się kurwa zapisuje na wf

5.1.10. Porównanie rezultatów użytych metod

Przy porównaniu wyników musimy zwrócić uwagę na sposób ewaluacji. Wszystkie powyższe metody były testowane na zbiorze treningowym składającym się z 945 wypowiedzi, kiedy oryginalnie było ich 1000. Usunięto 55 z nich, ponieważ były to retweety - tekst wypowiedzi był taki sam jak innej w zbiorze danych, jedynie był dodany znacznik „RT". Z uwagi na popularną praktykę usuwania zduplikowanych danych, wydaje się to być dobry krok ich przetwarzania. Jednak po dogłębnej analizie sposobu ewaluacji modeli uczestników konkursu okazuje się, że były one testowane na całym zbiorze 1000 tekstów. W takim wypadku wyniki naszych modeli są nieco niższe niż z poprzednim zbiorem danych. Przykładowo, najlepszych z kategorii: dla PRADO+LSH wynik spadł z $F1 = 60.4$ na $F1 = 59.0$, dla PRADO+FastText mamy spadek z $F1 = 60.0$ na $F1 = 58.1$, natomiast dla zwykłego PRADO wynik to $F1 = 51.1$, wcześniej $F1 = 52.1$. Relacje między badanymi metodami, a zarazem wnioski pozostają niezmienione i wciąż mamy model zajmujący pierwsze miejsce w konkursie.

Tablica 5.10: Porównanie wyników używanych metod, 1000 komentarzy w zbiorze testowym

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
PRADO	51.1	48.7	53.7	86.2	6
PRADO + LSH	57.0	**61.2**	**59.0**	88.6	**1**
PRADO + FastText	**58.8**	57.5	58.1	**88.9**	2

Tablica 5.11: Porównanie wyników używanych metod, 945 komentarzy w zbiorze testowym. Posortowane względem $F1$

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
PRADO + LSH	57.5	**63.6**	**60.4**	89.3	**1**
PRADO + FastText	60.5	59.5	60.0	**89.8**	**1**
PRADO + Word2Vec	59.4	60.3	59.8	89.6	**1**
PRADO	49.2	55.4	52.1	87.0	4
Regresja logistyczna	56.0	46.3	50.7	88.5	6
SVM	58.6	33.9	42.9	88.5	10
PRADO + GloVe	41.8	42.2	42.0	85.1	10
Ensemble drzew decyzyjnych	45.7	35.5	40.0	86.3	11
Drzewo decyzyjne	**81.5**	18.2	29.7	86.3	14
Naive Bayes	51.7	12.4	20.0	87.4	15

5.2. Metody uczestników konkursu

Warto spojrzeć na metody używane przez innych uczestników konkursu, aby porównać je z naszym podejściem.

- Zwycięzca konkursu - model n-waves ULMFiT to zmodyfikowany dla języka polskiego neuronowy model językowy Universal Language Model Fine-Tuning bazujący na idei uczenia transferowego. Opiera się on na sieci typu AWD-LSTM [46], który nie korzysta z mechanizmu uwagi, ale używa kilka różnych technik regularyzacji poprzez *Dropout*. Model był najpierw trenowany na korpusie z Reddita, a następnie krótko uczony na zbiorze właściwym do zadania w celu dostosowania wag sieci neuronowej.

- Przetak to system korzystający z *Morfeusza* i słownika wyrazów obelżywych, wulgarnych ubliżających i wulgarnych pochlebiających. Ponadto poprawia on literówki i zamiany znaków na inne. Stosuje on prosty model regresji logistycznej.

- Model ULMFiT + SentencePiece + BranchingAttention nie został opisany, ale z jego nazwy możemy wnioskować, że korzysta on z modelu językowego ULMFiT, algorytmu SentencePiece do podziału zdań na tokeny oraz mechanizmu uwagi.

- Trzy kolejne metody ensamble spacy + tpot + BERT, ensamble + fastai, ensenble spacy + tpot to propozycje jednego zespołu. Pierwsza z nich opiera się na modelu BERT, drugi to ponownie ULMFiT, a trzeci to model regresji logistycznej. Wszystkie 3 używały TPOT - biblioteki to optymalizacji przetwarzania cech i automatyzacji wyboru parametrów.

- Modele Rafal-1 i Rafal-2 korzystały z popularnych podejść takich jak n-gramy, sieci typu LSTM i GRU z zanurzeniami słów.
- Systemy model1-svm, model2-gru i model3-flair zostały zgłoszone przez jednego autora. Prosty model z SVM z TF-IDF, podwójnie skierowany GRU oraz model korzystający z biblioteki Flair oraz z zanurzeń kontekstowych.
- Model fasttext korzystał jak nazwa wskazuje z zanurzeń typu FastText.
- SCWAD-CB to model sieci neuronowej bazujący na zanurzeniach LASER.

Tablica 5.12: Porównanie wyników uczestników konkursu PolEval 2019, zadanie 6.1

Metoda	**Pr**	**Rc**	**F1**	**Acc**	**Pos**
n-waves ULMFiT	**66.7**	52.2	**58.6**	**90.1**	**1**
Przetak	66.3	51.5	58.0	90.0	2
ULMFiT + SentencePiece + BranchingAttention	52.9	54.5	53.7	87.4	3
ensamble spacy + tpot + BERT	52.7	50.7	51.7	87.3	4
ensamble + fastai	52.7	50.7	51.7	87.3	5
ensenble spacy + tpot	43.1	**58.2**	49.5	84.1	6
Rafal-1	41.1	56.7	47.6	83.3	7
Rafal-2	41.4	53.7	46.7	83.6	8
model1-svm	60.5	36.6	45.6	88.3	9
fasttext	58.1	32.1	41.3	87.8	10
SCWAD-CB	52.0	30.6	38.5	86.9	11
model2-gru	63.8	22.4	33.1	87.9	12
model3-flair	81.8	13.4	23.1	88.0	13
Task 6: Automatic cyberbullying detection (J.K.)	17.4	32.1	22.6	70.5	14

W powyższych rozwiązaniach pojawiają się pełne modele językowe, model bazujący na danych statystycznych, jak i kilka rozwiązań opartych na tych samym idea jak te pokazane w pracy - SVM, FastText czy zanurzenia. Warto wspomnieć, że w opisywanych metodach nie pojawia się próba skorzystania z emotikon czy z Locality-sensitive hashing, co czyni nasze podejście bardziej unikatowym. Wystąpienia PRADO w konkursie było niemożliwe, z uwagi na to praca na te temat tej architektury została opublikowana już po konkursie.

5.3. Wnioski

Przetestowaliśmy wiele metod klasycznego uczenia maszynowego, oryginalną implementację PRADO oraz jej modyfikacje. W opisywanym zadaniu 6.1 z konkursu PolEval już klasyczne algorytmy były w stanie osiągnąć przyzwoite wyniki.

Zbalansowane drzewa decyzyjne uzyskały $F1 = 40.0$, co jednak zostało zdecydowanie poprawione przez regresję logistyczną - $F1 = 50.7$. Już tutaj został użyty własny pomysł przedstawienia emotikon jako słów przedstawiających emocje, okazał się skuteczny w wielu przypadkach.

Oryginalna implementacja PRADO z projekcją osiągnęła $F1 = 52.1$, poprawiając wyniki wszystkich poprzednich metod. Również tutaj emotikony pomogły osiągnąć lepszy rezultat. Następnie, projekcja została zastąpiona przez zanurzenia słów typu Word2Vec. Uwagi w literaturze znalazły zastosowanie w praktyce – przy niewielkim zbiorze danych architektura skip-gram osiągnęła znacznie lepsze wyniki, $F1 = 59.8$. Taki wynik umożliwiłby zwycięstwo w konkursie PolEval.

Zanurzenia typu FastText zadziałały podobnie jak Word2Vec, okazuje się, że możliwość reprezentacji słów nieznanych wcześniej w korpusie może pomóc w wynikach. Tutaj uzyskaliśmy $F1 = 60.0$. Dodanie zbioru treningowego do korpusu okazało się dobrym pomysłem. Zanurzenia GloVe niestety nie poradziły sobie, uzyskując wyniki zbliżone do SVM. Bardzo dobrze wypadły zanurzenia oparte na idei LSH, budowane na podstawie wcześniej utworzonych zanurzeń typu FastText. Własna modyfikacja pozwoliła osiągnąć najlepszy wynik spośród wszystkich modeli - $F1 = 60.4$. Warto zwrócić uwagę na inny model typu LSH, który osiągnął Recall równy 68.6 i w pewnych zastosowaniach mógłby być preferowany. Ważną zaletą LSH jest możliwość jego efektywnego kodowania - przy tej samej długości wektorów możemy uzyskać 32-krotnie mniejsze zapotrzebowanie na pamięć.

Podsumowując, pokazaliśmy, że pomysł zastąpienia projekcji w architekturze PRADO klasycznymi zanurzeniami słów może dać znacząco lepsze rezultaty w zadaniu wykrywania mowy nienawiści.

5.4. Możliwe rozszerzenia

Powyższe eksperymenty i wyniki można rozszerzyć przez na co najmniej kilka sposobów. W pewnych przypadkach uwzględnienie interpunkcja pomogłaby uzyskać dodatkowe informacje o kontekście, więc jej odpowiednie przetworzenie i umieszczenie w zanurzeniach mogłoby polepszyć wyniki. Z uwagi na dobre zachowanie modeli po dodaniu danych treningowych i emotikon, adekwatnym pomysłem byłoby zebranie większej ilości surowych komentarzy i dołączenie ich do korpusu. Innym rozwinięciem byłoby zastosowanie zanurzeń kontekstowych, takich jak wyznaczane przez BERT czy ELMO. Kolejnym krokiem byłoby skorzystanie z większej mocy obliczeniowej i utworzenie dłuższych wektorów zanurzeń bądź skorzystanie z wektorów pretrenowanych. Poza zastosowaniem Morfeusza do szukania lematów słów, inne uwagi dotyczące przetwarzanie danych powinny mieć również zastosowanie w innych językach. W związku z tym w łatwy sposób można wypróbować pomysł modyfikacji PRADO z zanurzeniami w zadaniu wykrywania mowy nienawiści również w innych językach.

Bibliografia

[1] Alan Turing. "Computing machinery and intelligence". W: *Mind* LIX (1950), s. 433–460.

[2] J.J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities". W: *Proceedings of the National Academy of Sciences* 79.8 (1982), s. 2554–2558. DOI: 10.1073/pnas.79.8.2554.

[3] Jay L McClelland, David E Rumelhart i Geoffrey E Hinton. *The appeal of parallel distributed processing*. MIT Press, 1986.

[4] C.E. Shannon. "A Mathematical Theory of Communication". W: *The Bell System Technical Journal* XXVII (1948), 379–423 (July), 623–656 (October).

[5] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." W: *Psychological review* 65 6 (1958), s. 386–408.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent i Christian Jauvin. "A Neural Probabilistic Language Model". W: *Journal of Machine Learning Research* 3 (2003), s. 1137–1155.

[7] Alex Krizhevsky, Ilya Sutskever i Geoff Hinton. "ImageNet classification with deep convolutional neural networks". W: *NIPS*. 2012, s. 1106–1114.

[8] Instytut Języka Polskiego PAN. *Wielki słownik języka polskiego*. https://wsjp.pl/haslo/podglad/41203/mowa-nienawisci. [Online; accessed 25-January-2022]. 2021.

[9] Ziqi Zhang i Lei Luo. "Hate Speech Detection: A Solved Problem? The Challenging Case of Long Tail on Twitter". W: *CoRR* abs/1803.03662 (2018). arXiv: 1803.03662. URL: http://arxiv.org/abs/1803.03662.

[10] Ebru Aydoğan i M. Ali Akcayol. "A comprehensive survey for sentiment analysis tasks using machine learning techniques". W: *2016 International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)*. 2016, s. 1–7. DOI: 10.1109/INISTA.2016.7571856.

[11] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta i Vasudeva Varma. "Deep Learning for Hate Speech Detection in Tweets". W: *CoRR* abs/1706.00188 (2017). arXiv: 1706.00188. URL: http://arxiv.org/abs/1706.00188.

[12] Francisco Javier Ordóñez i Daniel Roggen. "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition". W: *Sensors* 16.1 (2016). ISSN: 1424-8220. DOI: 10.3390/s16010115. URL: https://www.mdpi.com/1424-8220/16/1/115.

[13] Pedro Alonso, Rajkumar Saini i György Kovács. "Hate Speech Detection Using Transformer Ensembles on the HASOC Dataset". W: wrz. 2020, s. 13–21. ISBN: 978-3-030-60275-8. DOI: 10.1007/978-3-030-60276-5_2.

[14] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy i Samuel R. Bowman. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". W: In the Proceedings of ICLR. 2019.

[15] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy i Samuel R. Bowman. "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems". W: *arXiv preprint 1905.00537* (2019).

[16] Piotr Rybak, Robert Mroczkowski, Janusz Tracz i Ireneusz Gawlik. "KLEJ: Comprehensive Benchmark for Polish Language Understanding". W: *CoRR* abs/2005.00630 (2020). arXiv: 2005.00630. URL: https://arxiv.org/abs/2005.00630.

[17] Maciej Ogrodniczuk i Łukasz Kobyliński, red. *Proceedings of the PolEval 2019 Workshop*. Warsaw, Poland: Institute of Computer Science, Polish Academy of Sciences, 2019. ISBN: 978-83-63159-28-3. URL: http://2019.poleval.pl/files/poleval2019.pdf.

[18] Bertie Vidgen i Leon Derczynski. "Directions in Abusive Language Training Data: Garbage In, Garbage Out". W: *CoRR* abs/2004.01670 (2020). arXiv: 2004.01670. URL: https://arxiv.org/abs/2004.01670.

[19] *International Workshop on Semantic Evaluation*. https://semeval.github.io/. [Online; accessed 26-January-2022]. 2021.

[20] *Evaluation of NLP and Speech Tools for Italian*. https://www.evalita.it/. [Online; accessed 26-January-2022]. 2021.

[21] Michal Ptaszynski i Fumito Masui. *Automatic Cyberbullying Detection: Emerging Research and Opportunities*. List. 2018. ISBN: 9781522552499. DOI: 10.4018/978-1-5225-5249-9.

[22] Prabhu Kaliamoorthi, Sujith Ravi i Zornitsa Kozareva. "PRADO: Projection Attention Networks for Document Classification On-Device". W: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, list. 2019, s. 5012–5021. DOI: 10.18653/v1/D19-1506. URL: https://aclanthology.org/D19-1506.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser i Illia Polosukhin. "Attention Is All You Need". W: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[24] Yann Lecun, Patrick Haffner i Y. Bengio. "Object Recognition with Gradient-Based Learning". W: (sierp. 2000).

[25] Sujith Ravi. "ProjectionNet: Learning Efficient On-Device Deep Networks Using Neural Projections". W: *CoRR* abs/1708.00630 (2017). arXiv: 1708.00630. URL: http://arxiv.org/abs/1708.00630.

[26] Sujith Ravi. "Efficient On-Device Models using Neural Projections". W: *Proceedings of the 36th International Conference on Machine Learning*. Red. Kamalika Chaudhuri i Ruslan Salakhutdinov. T. 97. Proceedings of Machine Learning Research. PMLR, 2019, s. 5370–5379. URL: https://proceedings.mlr.press/v97/ravi19a.html.

[27] Sujith Ravi i Zornitsa Kozareva. "Self-Governing Neural Networks for On-Device Short Text Classification". W: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, paź. 2018, s. 887–893. DOI: 10.18653/v1/D18-1105. URL: https://aclanthology.org/D18-1105.

[28] Tomas Mikolov, Kai Chen, Greg Corrado i Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space". W: *arXiv preprint arXiv:1301.3781* (2013).

[29] Piotr Bojanowski, Edouard Grave, Armand Joulin i Tomás Mikolov. "Enriching Word Vectors with Subword Information". W: *CoRR* abs/1607.04606 (2016). arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606.

[30] Fengfu Li i Bin Liu. "Ternary Weight Networks". W: *CoRR* abs/1605.04711 (2016). arXiv: 1605.04711. URL: http://arxiv.org/abs/1605.04711.

[31] Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas i Richard A. Harshman. "Indexing by latent semantic analysis". W: *JASIS* 41.6 (1990), s. 391–407.

[32] Y. Bengio i Yann Lecun. "Scaling learning algorithms towards AI". W: sty. 2007.

[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee i Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". W: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[34] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee i Luke Zettlemoyer. "Deep contextualized word representations". W: *CoRR* abs/1802.05365 (2018). arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365.

[35] Armand Joulin, Edouard Grave, Piotr Bojanowski i Tomás Mikolov. "Bag of Tricks for Efficient Text Classification". W: *CoRR* abs/1607.01759 (2016). arXiv: 1607.01759. URL: http://arxiv.org/abs/1607.01759.

[36] Jeffrey Pennington, Richard Socher i Christopher Manning. "GloVe: Global Vectors for Word Representation". W: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, paź. 2014, s. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[37] Aristides Gionis, Piotr Indyk i Rajeev Motwani. "Similarity Search in High Dimensions via Hashing". W: *Proceedings of the 25th International Conference on Very Large Data Bases*. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, s. 518–529. ISBN: 1558606157.

[38] Kristina Gligoric, Ashton Anderson i Robert West. "Adoption of Twitter's New Length Limit: Is 280 the New 140?" W: *CoRR* abs/2009.07661 (2020). arXiv: 2009.07661. URL: https://arxiv.org/abs/2009.07661.

[39] Marek Troszyński i Aleksander Wawer. "Czy komputer rozpozna hejtera? Wykorzystanie uczenia maszynowego (ML) w jakościowej analizie danych. [Can a Computer Recognize Hate Speech? Machine Learning (ML) in Qualitative Data Analysis]". W: *Przegląd Socjologii Jakościowej* XIII (maj 2017), s. 62–80.

[40] Mateusz Kostrzewski. *Polish Twitter Sentiments*. https://www.kaggle.com/mateuszkostrzewski/polish-twitter-sentiments/version/1. [Online; accessed 22-January-2022]. 2021.

[41] Jan Kocoń, Piotr Miłkowski i Monika Zaśko-Zielińska. "Multi-Level Sentiment Analysis of PolEmo 2.0: Extended Corpus of Multi-Domain Consumer Reviews". W: *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*. Hong Kong, China: Association for Computational Linguistics, list. 2019, s. 980–991. DOI: 10.18653/v1/K19-1092. URL: https://aclanthology.org/K19-1092.

[42] Jan Kocoń, Monika Zaśko-Zielińska, Piotr Miłkowski, Arkadiusz Janz i Maciej Piasecki. *Wroclaw Corpus of Consumer Reviews Sentiment (WCCRS)*. CLARIN-PL digital repository. 2019. URL: http://hdl.handle.net/11321/700.

[43] Witold Kieraś i Marcin Woliński. "Morfeusz 2 – analizator i generator fleksyjny dla języka polskiego". W: *Język Polski* XCVII.1 (2017), s. 75–83.

[44] Maciej Ogrodniczuk i Łukasz Kobyliński, red. *Proceedings of the PolEval 2018 Workshop*. Warsaw, Poland: Institute of Computer Science, Polish Academy of Sciences, 2018. ISBN: 978-83-63159-27-6. URL: http://2018.poleval.pl/files/poleval2018.pdf.

[45] Prabhu Kaliamoorthi i Yicheng Fan. *Sequence Projection Models*. `https://github.com/tensorflow/models/tree/master/research/seq_flow_lite`. [Online; accessed 26-January-2022]. 2021.

[46] Jeremy Howard i Sebastian Ruder. "Fine-tuned Language Models for Text Classification". W: *CoRR* abs/1801.06146 (2018). arXiv: `1801.06146`. URL: `http://arxiv.org/abs/1801.06146`.