Politechnika Łódzka
Wydział Elektrotechniki Elektroniki
Informatyki i Automatyki

inż. Marta Glanowska

Nr albumu: 244815

PRACA DYPLOMOWA MAGISTERSKA

na kierunku Sztuczna Inteligencja i Uczenie Maszynowe

Klasyfikacja typu osobowości na podstawie postów internetowych

Instytut Informatyki Stosowanej

Promotor: dr inż. Robert Susik

Łódź, 2024

Streszczenie

Nowoczesny świat podkreśla istotę stałego rozwoju osobistego, zachęca do świadomego poznawania swojego wnętrza i działających w nim schematów. Objawia się to między innymi w sferze biznesu, w której coraz częściej mówi się o umiejętnościach miękkich, a środki finansowe przekazywane są na rozwój naturalnych zdolności pracowników. Jednymi z dobrze osadzonych w tej rzeczywistości narzędzi są testy osobowości, których przeprowadzenie zajmuje jednak określoną ilość czasu. Dobrą alternatywą mógłby być system automatycznego rozpoznawania osobowości na podstawie innego rodzaju danych. W przedstawionej pracy zbadano możliwości zastosowania metod uczenia maszynowego i sztucznej inteligencji w problemie klasyfikacji szesnastu typów osobowości modelu MBTI. Podstawą modelowania była baza rzeczywistych wpisów zamieszczonych w Internecie. Okazały się one być wartościowym źródłem informacji o typach osobowości autorów. W toku badań przeprowadzono eksperymenty z użyciem czterech tradycyjnych algorytmów uczenia maszynowego oraz czterech architektur sieci neuronowych zbudowanych z użyciem nowoczesnych bloków przetwarzania tekstu. Ewaluacja wartości miary dokładności oraz wskaźnika F_1 zaproponowanych struktur jednoznacznie wyróżniła model XGBoost jako najskuteczniejsze rozwiązanie, o wynikach konkurencyjnych do innych obecnych w literaturze strategii. Rezultat przeprowadzonego badania potwierdza przydatność metod uczenia maszynowego na płaszczyźnie psychologicznej. Wskazuje potencjał narzędzi opartych na wytrenowanych modelach we wspieraniu specjalistów pracujących z testami osobowości podczas ich codziennych wyzwań.

Słowa kluczowe: uczenie maszynowe, sztuczna inteligencja, przetwarzanie języka naturalnego, klasyfikacja wieloklasowa, osobowość, model MBTI

Abstract

The modern world emphasizes the importance of constant personal development and encourages conscious exploration of one's interior and the patterns operating within it. This is manifested, among others, in the business sphere, where soft skills are increasingly talked about and financial resources are allocated to the development of employees' natural abilities. One of the tools well embedded in this reality are personality tests, which, however, take a certain amount of time to conduct. A good alternative could be an automatic personality recognition system based on other types of data. The presented work examines the possibilities of using machine learning and artificial intelligence methods in the problem of classifying sixteen personality types of the MBTI model. The basis for modeling was a database of actual entries posted on the Internet. They turned out to be a valuable source of information about the authors' personality types. During the research, experiments were carried out using four traditional machine learning algorithms and four neural network architectures built using modern text processing blocks. The evaluation of the value of the accuracy measure and the F_1 index of the proposed structures clearly distinguished the XGBoost model as the most effective solution, with results competitive to other strategies available in the literature. The result of the study confirms the usefulness of machine learning methods in the psychological field. It highlights the potential of tools based on trained models to support professionals working with personality tests in their everyday challenges.

Keywords: machine learning, artificial intelligence, natural language processing, multiclass classification, personality, MBTI model

Spis treści

Wykaz symboli i oznaczeń

O ile nie jest to inaczej wyszczególnione w tekście praca ta wykorzystuje poniższą notację:

- ln oznacza logarytm naturalny, czyli logarytm o podstawie e,
- exp oznacza funkcję eksponencjalną,
- $tanh$ oznacza tangens hiperboliczny,
- σ oznacza funkcję logistyczną,
- $p(y|x)$ oznacza prawdopodobieństwo warunkowe wystąpienia y pod warunkiem zajścia x,
- $|D|$ oznacza liczbę dokumentów w bazie,
- d_l oznacza dokument l bazy,
- K oznacza liczbę klas,
- m oznacza liczbę próbek w zbiorze treningowym,
- x oznacza wektor cech wejściowych,
- x_i oznacza przykład i danych wejściowych,
- x_t oznacza dane wejściowe w kroku t,
- y oznacza zmienną celu problemu klasyfikacyjnego,
- y_i^0 oznacza początkowe przypuszczenie dla przykładu x_i,
- y_i oznacza predykcję modelu dla przykładu x_i,
- y_t oznacza wynik modelu z kroku t,
- Y_i oznacza rzeczywistą etykietę przykładu x_i,
- θ oznacza wektor parametrów,
- θ_0 oznacza wyraz wolny,
- θ_k oznacza wektor parametrów modelu dla klasy k,
- ${\theta_k}^T$ oznacza transponowany wektor parametrów modelu dla klasy k,
- Θ oznacza macierz parametrów,
- λ oznacza parametr regularyzacji,
- C oznacza parametr regularyzacji definiowany jako odwrotność λ,
- ξ oznacza zmienną uzupełniającą,
- α oznacza parametr wygładzania addytywnego,
- I oznacza miarę zanieczyszczenia,
- D_p oznacza zestaw danych węzła nadrzędnego w drzewie,

- D_j oznacza zestaw danych j-tego węzła potomnego,
- $\frac{d}{df}$ oznacza pochodną pierwszego rzędu funkcji f zwaną gradientem,
- $\frac{d^2}{df^2}$ oznacza pochodną drugiego rzędu funkcji f zwaną hessianem,
- g_i oznacza gradient dla przykładu x_i,
- h_i oznacza hessian dla przykładu x_i,
- r_i oznacza rezyduum dla próbki x_i,
- γ oznacza parametr przycinania drzewa,
- W_{wyj} oznacza wartość wyjściową liścia w drzewie,
- h_t oznacza stan ukryty komórki LSTM w takcie t i pamięć komórki GRU w takcie t,
- c_t oznacza pamięć długoterminową komórki LSTM w takcie t,
- ϕ oznacza funkcję aktywacji neuronów komórki,
- i_t, f_t, o_t oznaczają wyjścia kolejno bramki wejściowej, zapominającej i wyjściowej komórki LSTM w takcie t
- z_t, r_t oznaczają wyjścia bramek komórki GRU w takcie t,
- g_t oznacza wyjście warstwy głównej w takcie t,
- W_x oznacza wagi wszystkich neuronów komórki dla danych wejściowych,
- W_{xi}, W_{xf}, W_{xo} oznaczają wagi neuronów dla danych wejściowych kolejno bramki wejściowej, zapominającej i wyjściowej komórki LSTM,
- W_{xz}, W_{xr} oznaczają wagi neuronów dla danych wejściowych bramek komórki GRU,
- W_{xg} oznacza wagi neuronów dla danych wejściowych warstwy głównej,
- W_y oznacza wagi wszystkich neuronów komórki dla danych wyjściowych poprzedniego taktu,
- W_{hi}, W_{hf}, W_{ho} oznaczają wagi neuronów dla stanu ukrytego kolejno bramki wejściowej, zapominającej i wyjściowej komórki LSTM,
- W_{hz}, W_{hr} oznaczają wagi neuronów dla stanu ukrytego bramek komórki GRU,
- W_{hg} oznacza wagi neuronów dla stanu ukrytego warstwy głównej,
- b oznacza obciążenia dla wszystkich neuronów komórki,
- b_i, b_f, b_o oznaczają obciążenia neuronów komórek kolejno bramki wejściowej, zapominającej i wyjściowej komórki LSTM,
- b_z, b_r oznaczają obciążenia neuronów bramek komórki GRU
- b_g oznacza obciążenia neuronów warstwy głównej,

- F_1 oznacza metrykę ewaluacji modelu będącą kombinacją precyzji i czułości,
- unigram oznacza n-gram jednoelementowy,
- bigram oznacza n-gram dwuelementowy,
- TF-IDF oznacza algorytm ważenia częstością termów (ang. *Term Frequency-Inverse Document Frequency*),
- GloVe oznacza jeden z algorytmów do tworzenia reprezentacji wektorowej słów (ang. *Global Vectors*),
- kNN oznacza algorytm k najbliższych sąsiadów (ang. *k-Nearest Neighbors*)
- SVM oznacza maszynę wektorów nośnych (ang. *Support Vector Machine*),
- XGBoost oznacza algorytm ekstremalnego gradientowego wzmocnienia drzew,
- ReLU oznacza funkcję rampy (ang. *rectified linear unit*),
- Adam oznacza jeden z algorytmów optymalizujących (ang. *Adaptive Moment Estimation*),
- LSTM oznacza alternatywę dla tradycyjnej komórki rekurencyjnej (ang. *Long Short-Term Memory*)
- GRU oznacza uproszczony wariant komórki LSTM (ang. *Gated Recurrent Unit*)
- BERT oznacza jeden z modeli językowych oparty na architekturze transformera (ang. *Bidirectional Encoder Representations from Transformers*)
- MBTI oznacza Myers-Briggs Type Indicator®.

1 Wprowadzenie

1.1 Motywacja

Kwestionariusz Myers-Briggs powstał w połowie ubiegłego wieku i jest jednym z najbardziej rozpowszechnionych modeli osobowości w biznesie[1], szczególnie w środowisku rekruterów, a także w szeroko pojętym doradztwie i coachingu. Przez lata był agresywnie reklamowany, aż na dobre osadził się w codziennych praktykach biznesowych, a także przyjął się w kulturze masowej jako sposób definicji i opisu natury poszczególnych jednostek. Mimo wątpliwości specjalistów[2] co do jego skuteczności, model ten wciąż znajduje szerokie zastosowanie we współczesnym świecie, gdyż bardzo często badani wspomnianym testem deklarują, że odnajdują się w opisie osobowości, do którego zostali zaklasyfikowali, tym samym czują się rozumiani[3]. Tagi definiujące użytkownika z użyciem typów MBTI pojawiły się nawet w niektórych społecznościach platformy Reddit[4], a także zostały wprowadzone do popularnych aplikacji randkowych, takich jak Tinder lub Boo, w celu poszukiwania jeszcze większej kompatybilności między potencjalnymi partnerami. Jedna z najbardziej popularnych stron internetowych[5] oferujących darmowy kwestionariusz Myers-Briggs jest łatwo dostępna, atrakcyjna wizualnie, a ponadto oferuje test w wielu językach, co daje wspomnianemu modelowi znaczną przewagę nad innymi testami osobowości.

W tej pracy postawiono pytanie badawcze, czy przy użyciu modelu sztucznej inteligencji można ułatwić proces definiowania typu osobowości, oszczędzając dzięki temu czas poświęcany na wykonanie standardowego kwestionariusza. Zmiana źródła danych z odpowiedzi zaznaczanych w teście na treść postów internetowych ma szczególną zaletę – ograniczenie lustrzanego efektu, kiedy opis danego typu zawiera informacje pochodzące z zaznaczonych odpowiedzi, przez co wydaje się być jeszcze wierniejszy. Jest to istotny aspekt, zwłaszcza że wyniki testu potrafią wahać się w zależności od okoliczności i samopoczucia danej

[1] . https://www.psychologytoday.com/us/blog/credit-and-blame-at-work/200806/the-use-and-misuse-of-personality-tests-for-coaching-and, dostęp z dnia 24.11.2023 r

[2] https://www.recruiter.com/recruiting/critique-of-the-myers-briggs-type-indicator-critique/, dostęp z dnia 24.11.2023 r.

[3] https://www.wired.com/story/myers-briggs-test-internet-fans/, dostęp z dnia 24.11.2023 r.

[4] https://www.reddit.com/

[5] https://www.16personalities.com/pl

osoby w chwili wykonywania kwestionariusza[6]. Z drugiej strony, brak dostępu do wpisów publikowanych w Internecie przez badaną osobę skutecznie uniemożliwia zastosowanie do klasyfikacji takiej innowacyjnej strategii. Można zastanowić się, czy w takim przypadku problem mógłby zostać rozwiązany zastępczo za pomocą innych próbek tekstu, na przykład wiadomości z konwersacji na komunikatorach. To zagadnienie nie mieści się jednak w zakresie przedstawionej pracy, byłoby natomiast ciekawym pomysłem na jej rozwinięcie. Badacze podejmujący się rozwiązania problemu klasyfikacji szesnastu osobowości MBTI często decydują się na opisanie go czterema osobnymi predykcjami binarnymi zamiast zbudowania modelu wieloklasowego. Dodatkowym pytaniem badawczym staje się, czy możliwe jest uzyskanie podobnych lub lepszych wyników do nich stosując drugie z tych podejść.

Ta praca ma szansę przysłużyć się do pełnego wykorzystania potencjału modelu MBTI. Skuteczność eksperymentu oznaczała zwiększenie autentyczności wyników wraz ze spotęgowaniem ich stałości, przy większej tolerancji na czynniki zewnętrzne wpływające na daną jednostkę.

Stworzony model sztucznej inteligencji może zostać wprowadzony do systemów rekrutacyjnych, by przeprowadzać wstępną ocenę osobowości kandydata na podstawie próbek tekstu, przykładowo pozyskanych z jego profesjonalnego profilu zawodowego w specjalistycznym serwisie społecznościowym, takim jak LinkedIn[7]. Takie rozpoznanie może zostać potwierdzone przez kwestionariusz na dalszym etapie rekrutacji, oszczędzając jednak czas obu stronom we wstępnym jej stadium. Należy jednak pamiętać, że kwestionariusze osobowości, chociaż mogą być pomocne w scharakteryzowaniu danego kandydata, nie powinny być czynnikiem kluczowym dla podejmowania decyzji o zatrudnieniu.

Model może być także wsparciem dla początkujących przedsiębiorców lub pracowników zastanawiających się nad swoimi naturalnymi rolami w grupie, a także rozpoznającymi swoje mocne i słabe strony. System może również zaistnieć jako wtyczka do popularnych portali społecznościowych, na których ludzie przez lata generują dziesiątki, a nawet setki postów. Takie narzędzie samorozwojowe za sprawą jednego kliknięcia byłoby w stanie określić typ osobowości danego użytkownika, a tym samym przybliżyć mu informacje na temat jego

[6] https://www.theguardian.com/science/brain-flapping/2013/mar/19/myers-briggs-test-unscientific, dostęp z dnia 24.11.2023 r.

[7] https://www.linkedin.com/

codziennych motywacji, czy sposobu komunikacji, a więc przysłużyć się do wykorzystania jego potencjału, a ponadto poprawy jego relacji z bliskimi ludźmi.

1.2 Cel i zakres pracy

Celem pracy jest eksploracja możliwości zastosowania metod uczenia maszynowego i sztucznej inteligencji w klasyfikacji typów osobowości na podstawie postów internetowych. Zakres pracy obejmuje przegląd literatury w tej tematyce, wstępne przetworzenie danych tekstowych, przygotowanie stosownych do specyfiki danych modeli reprezentacji numerycznych badanych fragmentów, implementację i trening wybranych klasyfikatorów oraz analizę uzyskanych wyników.

1.3 Model osobowości MBTI

Powstanie testu MBTI ma swoje korzenie w teorii osobowości szwajcarskiego psychiatry i psychologa Carla Gustava Junga (Jung, 2013). Jego koncepcja została rozwinięta w pierwszej połowie XX wieku przez amerykańską pisarkę i badaczkę Katharine Cook Briggs oraz jej córkę Isabel Briggs Myers, a dwadzieścia lat ich obserwacji doprowadziły ostatecznie do powstania kwestionariusza MBTI. Rzetelność i ważność modelu zostały potwierdzone i opisane w raporcie The Myers-Briggs Company[8].

Klasyfikacja MBTI opiera się na określeniu wartości dla czterech par kategorii[9], opisanych w Tabeli 1.

Tabela 1 – Opis par kategorii modelu MBTI. Źródło: opracowanie własne

	Para kategorii	
	Ekstrawersja (E)	**Introwersja (I)**
Poziom 1 – Źródło czerpania energii i komunikacja.	Koncentracja na świecie zewnętrznym. Komunikacja poprzez rozmowę. Towarzyskość.	Koncentracja na świecie wewnętrznym. Komunikacja poprzez pismo. Powściągliwość i nieufność.

[8] Na podstawie raportu https://eu.themyersbriggs.com/-/media/Files/PDFs/Technical-information/MBTI_reliability_and_validity_info.pdf

[9] Charakterystyki w *Tabeli 1* są skrótem bardziej szczegółowych opisów przedstawionych w artykule https://potencjalosobowosci.com/mbti/, dostęp z dnia 01.02.2024 r.

Poziom 2 – Sposób zbierania informacji i operowania nimi.	**Poznanie (S)** Zorientowanie na fakty. Rzeczowość. Zapamiętywanie szczegółów. Praktyka i doświadczenie.	**Intuicja (N)** Zorientowanie na możliwości. Kreatywność. Zapamiętywanie ogólnego obrazu. Analiza teoretyczna i inspiracja.
Poziom 3 – Motywacja podejmowania decyzji.	**Myślenie (T)** Rozum ponad serce. Rachunek „za i przeciw". Sprawiedliwość i logika. Skupienie na osiągnięciu celu.	**Odczuwanie (F)** Serce ponad rozum. Empatia i harmonia w grupie. Skupienie na byciu docenianym przez innych.
Poziom 4 – Styl życia i pracy.	**Osądzanie (J)** Systematyczność. Planowanie. Sztywne ramy.	**Obserwacja (P)** Spontaniczność. Swoboda. Otwartość na modyfikacje.

Istnieje 16 typów osobowości modelu MBTI. Czteroelementowa nazwa każdego typu jest prostym złożeniem liter z każdej z czterech par, która bardziej odpowiada naturze danej jednostki. W ten sposób, osoba o usposobieniu introwertyka (I), zorientowana na fakty i rzeczowość (S), kierująca się empatią i uczuciami (F), charakteryzująca się swobodnym stylem życia i elastycznością (P) najprawdopodobniej zostanie zaklasyfikowana do typu ISFP.

Czasami stosuje się koncentrację 16 typów w cztery grupy, aby sprawniej zorientować się w charakterystyce ich natury i pełnionych naturalnie funkcjach, są to: dążący do celu i pomysłowi Analitycy (INTJ, INTP, ENTP, ENTJ), kreatywni i zorientowani na ludzi Dyplomaci (INFJ, INFP, ENFJ, ENFP), praktyczni i skuteczni Strażnicy (ISTJ, ISFJ, ESTJ, ESFJ) oraz otwarci i spontaniczni Odkrywcy (ISTP, ISFP, ESTP, ESFP).

Wadą systemu MBTI niewątpliwie jest brak wartości pośrednich dla poszczególnych kategorii. Na przykład, w tej klasyfikacji badany nie może zostać uznany za ambiwertyka, czyli osobę posiadającą cechy z pogranicza ekstrawersji i introwersji. Może być to problematyczne w przypadku osób „uniwersalnych", u których przychylność do jednego z biegunów w parze kategorii nie jest ewidentna. Tacy ludzie nierzadko są postrzegani jako elastyczni, potrafiący dostosować się do sytuacji, korzystający z różnych swoich atrybutów zależnie od potrzeb i okoliczności. Wykonanie wspomnianego testu może okazać się dla nich frustrującym

doświadczeniem, ponieważ oscylując pośrodku dwóch biegunów kategorii, wynik klasyfikacji jest narażony na dużą zmienność wraz z kolejnymi wykonaniami testu.

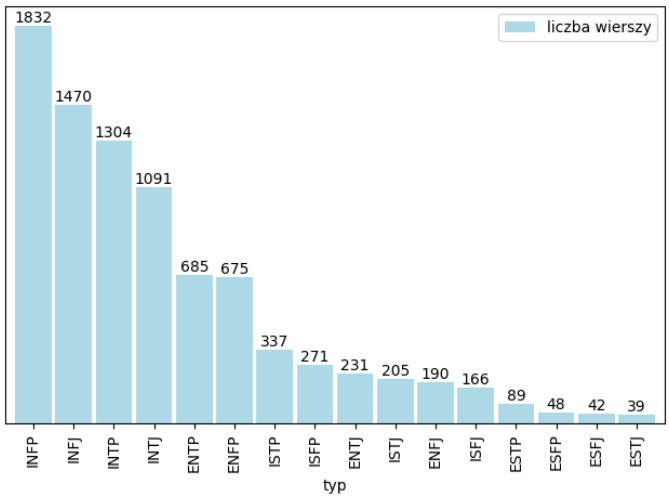
1.4 Zbiór danych

(MBTI) Myers-Briggs Personality Type Dataset to zbiór danych dostępny publicznie na platformie Kaggle[10]. Zawiera on 8675 wierszy, przy czym każdy z nich odpowiada ostatnim pięćdziesięciu postom udostępnionym przez danego użytkownika na forum PersonalityCafe[11] wraz z charakteryzującym go typem osobowości w konwencji modelu MBTI. Zbiór odznacza się różnorodnością w kontekście opisywanych przez niego osób, a także samego stylu pisania. Zebranie danych z istniejącego forum, a nie w sztucznych warunkach, przyczyniło się do powstania próbek, które można określić jako autentyczne i odwzorowujące realne życie.

Rys. 1 przedstawia dystrybucję próbek między klasami. Każdy z wierszy zawiera maksymalnie pięćdziesiąt postów internetowych. Zbiór nie jest zbalansowany – zebrano ponad czterdziestokrotnie więcej informacji na temat osób o typie INFP niż osób scharakteryzowanych jako ESTJ czy ESFJ. Cztery najwyższe słupki wykresu opisują liczbę postów należących do osób określonych jako introwertyczne. Ponadto, około 85% danych odnosi się do użytkowników forum o typie osobowości bazującym na intuicji (składowa N), a nie zmysłach (składowa S).

[10] https://www.kaggle.com/datasets/datasnaek/mbti-type
[11] https://www.personalitycafe.com/

Rys. 1 – Dystrybucja próbek między klasami. Źródło: opracowanie własne



2 Przegląd literatury

Klasyfikacja typów osobowości modelu MBTI na podstawie próbek tekstu przy użyciu uczenia maszynowego jest zagadnieniem chętnie podejmowanym przez badaczy na przestrzeni ostatnich lat.

W 2017 roku Rayne Hernandez i Ian Scott Knight opisali problem badawczy w swoim artykule (Hernandez & Knight, 2017) i odnotowali, że istniejące ówcześnie rozwiązania są nieliczne. Ich praca jako jedna z pierwszych bazowała na zbiorze *(MBTI) Myers-Briggs Personality Type Dataset.* Posty internetowe zostały wstępnie przetworzone, poszczególne słowa zakodowane w przestrzeni 50-wymiarowej za pomocą algorytmu GloVe, a następnie wymodelowane za pomocą rekurencyjnych sieci neuronowych z komórkami LSTM, osobno dla każdej pary kategorii. Wspomniany artykuł dał podwaliny kolejnym pracom i okazał się jednym z najczęściej cytowanych w późniejszych latach.

We wspomnianym roku, próbę modelowania dokładnie tego samego zbioru danych podjęli również Brandon Cui i Calvin Qi (Cui & Qi, 2017). Ich działania były szerzej zakrojone – zajęli się zarówno problemem klasyfikacji szesnastoklasowej jak i tworzeniem czterech odrębnych klasyfikatorów binarnych odpowiedzialnych za kolejne pary kategorii. Autorzy postanowili skupić się na drugim z wymienionych podejść, deklarując niskie wartości dokładności klasyfikatorów wieloklasowych, wynoszących poniżej 25%. W tej pracy dane tekstowe zostały wyrażone numerycznie w głównej mierze za pomocą tradycyjnej techniki worka słów (ang. *Bag of Words*).

Bianca Antonio wraz z zespołem wykonali w 2018 roku wnikliwy projekt (Antonio, et al., 2018), w którym udało im się wyczerpująco opisać sam zbiór danych, przetestować kilka metod próbkowania w celu zniwelowania braku zbalansowania na przestrzeni szesnastu klas, przeprowadzić szczegółową ekstrakcję cech, włączając zliczanie wystąpień poszczególnych elementów postów, analizę sentymentu i otagowanie części mowy. Podobnie jak w poprzednich pracach, autorzy nie zdecydowali się jednak na modelowanie wieloklasowe. Po treningu modeli regresji logistycznej, maszyn wektorów nośnych (ang. *Support Vector Machines*), ekstremalnego wzmocnienia gradientowego (ang. *Extreme Gradient Boosting*) oraz lasów losowych, te ostatnie osiągnęły najwyższy iloczyn dokładności dla czterech modeli

binarnych, jednak wynik ten był niewiele lepszy od pracy Brandona Cui i Calvina Qi (Cui & Qi, 2017).

Podejście zaproponowane w 2020 roku przez inną grupę badaczy (Abidin, et al., 2020) zawierało porównanie skuteczności maszyn wektorów nośnych, modeli regresji logistycznej, algorytmów k najbliższych sąsiadów (ang. *k-Nearest Neighbors*) oraz lasów losowych bazując na danych zakodowanych za pomocą techniki Word2Vec, oraz dodatkowych cechach wybranych po analizie wskaźników korelacji Pearsona. W tym przypadku, las losowy osiągnął 100% dokładności, jednak również był trenowany odrębnie dla każdej pary kategorii.

Interesujące wnioski wysnuli Amirhosseini i Kazemian (Amirhosseini & Kazemian, 2020), udowadniając, że metoda ekstremalnego wzmocnienia gradientowego może być skuteczniejsza niż rekurencyjna sieć neuronowa w rozwiązaniu zadanego problemu. Istotnie, cząstkowe dokładności ich modeli były wyższe niż tych w artykule z 2017 roku (Hernandez & Knight, 2017).

Jedna z nowszych prac (Ontoum & Chan, 2022) podejmująca wspomniany problem badawczy odznacza się wyższą dokładnością klasyfikacji próbek tekstowych niż wymienione powyżej. W 2022 roku zespołowi badaczy w Tajlandii udało się osiągnąć ponad 40% dokładności zarówno dla naiwnych klasyfikatorów bayesowskich (ang. *Naive Bayes*) jak i maszyn wektorów nośnych. Przygotowanie danych do tych klasyfikatorów obejmowało zastosowanie techniki worka słów oraz ważenia częstością termów (ang. *Term Frequency-Inverse Document Frequency*). Ponadto, zaproponowana przez autorów dwukierunkowa sieć rekurencyjna z użyciem komórek LSTM osiągnęła niemal 50% skuteczności dla danych zakodowanych za pomocą tokenizera biblioteki *Keras*.

3 Wstęp teoretyczny

3.1 Przetwarzanie języka naturalnego

Przetwarzanie języka naturalnego (ang. *Natural Language Processing*) to dziedzina zajmująca się zastosowaniem technik analizy danych oraz modeli sztucznej inteligencji w celu generowania i rozumienia, a także rozwiązywania innych podobnych zadań opartych na tekście pisanym i mowie ludzkiej. Ta prężnie rozwijająca się w dzisiejszych czasach gałąź informatyki jest używana między innymi w systemach umożliwiających odczytanie tekstu ludzkim głosem, lub odwrotnie – narzędziach umożliwiającym głosowe wprowadzanie tekstu. Ma swoje odzwierciedlenie również w pracy wirtualnych doradców w postaci chatów na witrynach internetowych lub asystentów podczas rozmów telefonicznych z punktami usługowymi. Ponadto, jest ceniona jako fundamentalna składowa internetowych tłumaczy języków obcych, narzędzi do streszczania lub klasyfikacji dokumentów. Innym jej zastosowaniem może być automatyczna analiza sentymentu opinii konsumentów na forum.

3.2 Przygotowanie danych tekstowych

Tekst jest wymagającym źródłem danych. Zrozumienie kontekstu słów wymaga przeanalizowania nie tylko ich kolejności i sąsiedztwa innych wyrazów w zdaniach, ale także sytuacji, w jakiej wypowiedź się pojawiła. Należy pamiętać, że w języku naturalnym występowanie żartów sytuacyjnych, ironii czy związków frazeologicznych jest zjawiskiem codziennym, co oprócz ubarwienia języka skutkuje też niemożnością jednoznacznego zdefiniowania znaczenia poszczególnych słów.

Celem wstępnego przetwarzania tekstu jest sprowadzenie go do zwartej formy zawierającej maksymalną ilość kluczowych informacji, przy czym w zależności od rozwiązywanego problemu zestaw najbardziej akcentowanych cech może być inny.

Jedną z podstawowych metod obróbki tekstu jest tokenizacja (ang. *tokenization*). Polega ona na rozbiciu ciągłego tekstu na sekwencję tokenów, czyli mniejszych części. Często stosuje się podział na słowa, znaki, lub zdania. Równocześnie odrzuca się wybrane znaki interpunkcyjne.

W przygotowaniu tekstu istotne może być również zmniejszenie wszystkich liter – taki zabieg ma dwie zalety. Po pierwsze, automatycznie uszczupla rozmiar słownika. Ponadto,

wskazuje modelowi zgodność dwóch wystąpień tego samego słowa, spośród których jedno pojawiło się na początku zdania, a więc było pisane wielką literą. Z drugiej strony, może to również generować problemy, na przykład w przypadku wyrazu „Jagoda", gdzie wielkość litery wskazuje na fakt, że mowa jest o pewnej osobie, a nie o owocu.

Eliminacja słów, które same w sobie nic nie znaczą (ang. *stopwords*) to kolejna popularna technika wstępnego przetwarzania tekstu. Istnieją zaimki, spójniki, partykuły i inne wyrazy, które występują w języku pisanym nieraz częściej niż pozostałe słowa, równocześnie nie niosąc ze sobą znaczącej informacji. Świadczy o tym fakt, że są one pomijane przez wyszukiwarki. Takie wyrazy usuwa się zazwyczaj przygotowując tekst do klasyfikacji, z kolei nie jest to zalecane w pracy z generatywnymi modelami uczenia maszynowego, ponieważ tworzone przez nie wypowiedzi nie będą brzmieć naturalnie. Listę usuwanych słów należy jednak zweryfikować, gdyż niektóre standardowo na niej umieszczane wyrazy mogą okazać się znaczące w kontekście specyficznych zadań – na przykład badania częstości występowania zaimków „ja" i „moje" w wykrywaniu osobowości narcystycznych.

Innym prostym pomysłem jest praca nad poszczególnymi elementami tekstu, na przykład usuwanie linków czy emotikon, oflagowywanie części mowy, wydobywanie ze słów ich rdzeni (ang. *stemming*) lub lematów (ang. *lemmatization*).

Wybór technik różni się od specyfiki problemu, nie istnieje jeden uniwersalny pomysł na przygotowanie danych tekstowych.

3.3 Ekstrakcja dodatkowych cech

Oprócz modelowania samej wstępnie przetworzonej treści danego tekstu czy jego wycinków, można dodatkowo zdefiniować cechy opisujące numerycznie poszczególne próbki. Zależnie od problemu, inne charakterystyki mogą być znaczące. Fragment tekstu można zdefiniować zliczając występujące w nim poszczególne części mowy, wyznaczając jego długość bezwzględną lub mierzoną ilością zdań, przeprowadzając analizę sentymentu i przypisując wartość binarną, czy oznaczając emocje wiązane z poszczególnymi zwrotami.

3.4 Reprezentacja numeryczna tekstu

Wyrażanie tekstu liczbowo jest jedną z kluczowych operacji przetwarzania wstępnego ze względu na to, że modele uczenia maszynowego są trenowane właśnie na reprezentacji

numerycznej. Większość algorytmów opisanych poniżej jest dodatkowo specyficznymi ekstraktorami cech, ponieważ pozyskują one informacje na temat współwystępowania słów i ich częstości lub o kontekście.

3.4.1 Metody uwzględniające kolejność wyrazów

Najbardziej podstawową techniką kodowania tekstu jest stworzenie słownika istniejących w danym zbiorze danych wyrazów, a następnie zamiana słów w próbkach według jego indeksów. W zależności od potrzeb, wielkość słownika można ograniczyć. Uzyskany ciąg liczb całkowitych reprezentujących poszczególne słowa nazywany jest ciągiem tokenów.

Mechanizm mapowania wyrazów zakodowanych tokenami na wielowymiarową reprezentację wektorową (ang. *embedding*) jest natomiast pomysłem na rozszerzenie powyższej techniki o pewne informacje semantyczne. Ta technika ma dwie wyraźne zalety. Po pierwsze, można zauważyć, że w celu stworzenia binarnej reprezentacji próbek, zastosowanie kodowania one-hot przeprowadzonego na tokenach, mając na uwadze, że dla wielu tekstowych baz danych jest ich dziesiątki lub setki tysięcy, nie będzie optymalnym działaniem. Z pomocą przychodzi wówczas umiejętność przetworzenia informacji zawartej w tokenach w taki sposób, aby każde słowo zostało wyrażone jako punkt w przestrzeni o wcześniej jasno ustalonej, o wiele niższej liczbie wymiarów. Zmniejsza to ilość mocy obliczeniowej potrzebnej do samego wytrenowania modelu. Druga zaleta takiego podejścia, to możliwość uchwycenia w reprezentacji relacji zachodzących między wyrazami. Dzięki zrozumieniu kontekstu, słowa o podobnym znaczeniu semantycznym znajdują się w przestrzeni blisko siebie[12].

Do tworzenia reprezentacji numerycznej słów można również zastosować pretrenowany model typu BERT (Devlin, et al., 2018) będący transformerem zawierającym w swojej architekturze jedynie koder, z pominięciem dekodera. W tym przypadku w reprezentacji wektorowej poszczególnych słów kodowane są nie tylko podstawowe informacje o ich tokenach (ang. *token embeddings*), ale również oznaczenia fragmentów tekstu, z których pochodzą, w przypadku gdy zawierająca je próbka jest złączeniem kilku (ang. *segment embeddings*), a także ich pozycje w zawierających je próbkach (ang. *position embeddings*).

[12] https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce, dostęp z dnia 21.02.2024 r.

Powyższe metody są skuteczne w połączeniu z zastosowaniem rekurencyjnych sieci neuronowych, które następnie analizują otrzymane reprezentacje, a mając możliwość wychwycenia informacji o kolejności elementów sekwencji, badają strukturę zdań.

3.4.2 Metody statystyczne

Inną często stosowaną metodą zamiany tekstu na formę liczbową jest technika worka słów (ang. *Bag of Words*) budująca uproszczoną reprezentację próbki na podstawie częstości występowania w niej poszczególnych wyrazów, jednak w odróżnieniu do poprzednio opisanych technik, nie uwzględnia ona ich kolejności. Z tego powodu nie jest odpowiednia, aby przygotować dane wejściowe do sieci rekurencyjnych i łączy się ją z tradycyjnymi modelami uczenia maszynowego.

Rozwinięciem idei worka słów jest koncepcja ważenia częstością termów, bardziej znana jako TF-IDF. To metoda statystyczna, w której kluczową rolę grają dwa czynniki – częstość występowania danego słowa w konkretnej próbce, a także jego ważność, czyli frekwencja występowania na przestrzeni wszystkich próbek. Jest to forma normalizacji wyników, dzięki której dłuższe dokumenty nie mają większego wpływu na wartość docelową niż krótsze, a dodatkowo słowa występujące w każdym dokumencie są mało znaczące dla wyników, gdyż nie są wówczas cechą charakterystyczną właściwie żadnego z nich. Wyznaczenie reprezentacji należy rozpocząć od ustalenia, w ilu dokumentach d dane słowo (term) w_t się znalazło, a następnie podzielenia przez tą wartość liczby dokumentów w bazie $|D|$. Nałożenie logarytmu naturalnego gwarantuje kompresję różnicy między wynikami większymi niż jeden i zwiększenie dysproporcji dla wartości z przedziału $[0, 1]$. Następnie, należy pomnożyć częstości wystąpienia unikalnych wyrazów w danym dokumencie przez uzyskane wcześniej odpowiadające im wagi[13] (1). W poniższym wzorze składowa $n_{u,l}$ określa liczbę wystąpień konkretnego termu w_t w próbce d_l. W mianowniku tego ułamka znajduje się natomiast suma wystąpień wszystkich wyrazów w danym dokumencie d_l.

$$(TF - IDF)_{t,l} = TF_{t,l} * IDF_t = \frac{n_{t,l}}{\sum_u n_{u,l}} * ln \frac{|D|}{|\{d: w_t \epsilon\ d\}|} \quad (1)$$

Z komentarzem [MG1]: Wrócić tu

Można zażądać, aby łączne wartości słów w dokumentach były takie same. Dzięki temu, dłuższe dokumenty nie będą miały większego wpływu na wartość docelową niż krótsze.

[13] https://pl.wikipedia.org/wiki/TFIDF, dostęp z dnia 24.03.2024 r.

Implementacja takiego warunku odbywa się poprzez nałożenie wybranej normy na pierwszy z czynników. Dokumenty będą wówczas różnicowane według proporcji wag rozdzielonych między słowa, a nie ich łącznych wartości (Fenner, 2020, pp. 482-487).

Algorytm TF-IDF nadaje wagi słowom – największą wartość przypisuje rzadkim wyrazom, które w danej próbce wystąpiły często. Dopuszcza również traktowanie par, trójek lub większych zestawień słów, tak zwanych n-gramów, jako rekordów słownika. To oznacza, że dla każdego n-gramu liczona jest wówczas pojedyncza reprezentacja numeryczna, mimo że składa się on z n słów.

3.5 Klasyczne modele uczenia maszynowego

3.5.1 Wielomianowa regresja logistyczna

Klasyczna regresja logistyczna (ang. *logistic regression*) to procedura dostosowana do problemów klasyfikacyjnych natury binarnej poprzez szacowanie prawdopodobieństwa należenia próbki do pozytywnej klasy. W celu jego obliczenia wylicza się ważoną sumę cech wejściowych ujętą w funkcję logistyczną (2).

$$\hat{p} = \sigma(\theta^T \cdot x) \tag{2}$$

Funkcja logistyczna (ang. *logistic function*) jest szeroko stosowaną w statystyce funkcją sigmoidalną, która przekształca dane wejściowe do wartości z przedziału $(0, 1)$. Można ją wyrazić prostym równaniem (3).

Z komentarzem [MG2]: Konfrontacja z funkcją logitową?

$$\sigma(t) = \frac{1}{1 + exp(-t)} \tag{3}$$

Jeżeli wartość prawdopodobieństwa przynależności próbki do pozytywnej klasy jest równa lub przekracza 50%, model klasyfikuje ją do wspomnianej klasy zwracając wartość $y = 1$, natomiast w przeciwnym wypadku przypisuje etykietę klasy negatywnej $y = 0$.

Uczenie modelu polega na znalezieniu takiego wektora parametrów θ, aby zwracane wartości były wysokie dla przykładów klasy pozytywnej, a niskie dla próbek klasy negatywnej. Odwrotne przypisania powinny być karane podczas treningu. Umożliwia to logarytmiczna funkcja straty (ang. *log loss*) (4). Przyjęto oznaczenie m jako liczbę próbek w zbiorze treningowym.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i\, ln(\hat{p}_i) + (1 - y_i)\, ln(1 - \hat{p}_i)]$$

(4)

Z komentarzem [MG3]: Chodzi o proporcje, więc baza logarytmu nie ma tu znaczenia

Minimalizację tej funkcji kosztu można przeprowadzić między innymi za pomocą metody wsadowego gradientu prostego operując na pochodnych cząstkowych obliczanych dla każdego parametru θ_j (Géron, 2018, pp. 145-147).

W problemach wieloklasowych stosuje się wielomianowy wariant regresji logistycznej. W tym celu dla danej próbki należy najpierw obliczyć osobny wynik dla każdej klasy k uwzględniając jej indywidualny wektor parametrów θ_k (5).

$$s_k(x) = {\theta_k}^T \cdot x$$

(5)

Kolejnym krokiem jest przekazanie wyników wszystkich K klas dla danego przykładu do funkcji Softmax (6), która na tej podstawie oblicza prawdopodobieństwa przynależności do każdej z klas.

$$\hat{p}_k = \frac{exp\big(s_k(x)\big)}{\sum_{j=1}^{K} exp\left(s_j(x)\right)}$$

(6)

Predykcja $\hat{y}$ wielomianowego modelu regresji logistycznej to klasa, dla której wartość prawdopodobieństwa jest najwyższa. Z tego powodu, stosowanie tego algorytmu jest dobrym wyborem jedynie jeśli w danym zagadnieniu badawczym klasy wzajemnie się wykluczają.

Trenując ten wariant modelu, minimalizacji poddaje się nieco inną funkcję kosztu niż w problemie binarnym. Funkcja ta nazywana jest entropią krzyżową (7) i stosuje się ją często w uczeniu modeli przeznaczonych do klasyfikacji wieloklasowej. Jako argument przyjmuje macierz parametrów modelu Θ. Jeśli jednak dostępne są dwie klasy, warto zwrócić uwagę, że funkcja ta przekształca się w logarytmiczną funkcję straty (4). Podobnie jak w przypadku klasyfikacji binarnej, minimalizacja funkcji kosztu polega na użyciu metody wsadowego gradientu prostego, tuż po policzeniu wektora gradientów entropii krzyżowej dla każdej klasy w celu znalezienia najlepszej kombinacji parametrów macierzy Θ (Géron, 2018, pp. 150-152).

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_{i,k}\, ln\big(\hat{p}_{i,k}\big)$$

(7)

Z komentarzem [MG4]: Tu też nie ma znaczenia (e albo 2 jako podstawa), sprawdzić w książce

Z komentarzem [MG5R4]: To samo we wzorze 9

Model o zbyt dużej złożoności nie jest w stanie uogólniać informacji o klasach, natomiast model zbyt prosty nie potrafi wychwycić ich wyraźnych charakterystyk. Częstym zabiegiem

zapobiegającym nadmiernemu lub zbyt małemu dopasowaniu modelu do danych jest regularyzacja. Wariant L2 (8) karci duże wartości wag poprzez dodanie dodatkowego obciążenia do funkcji kosztu. Warto podkreślić, że kary nie są nakładane na wyrazy wolne Θ_0, ponieważ suma liczona jest od $j = 1$.

$$\frac{\lambda}{2}\|\theta\|^2 = \frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2$$

(8)

Implementacja regularyzacji polega na dodaniu członu regularyzacji do funkcji kosztu (9). W zaprezentowanych równaniach λ oznacza parametr regularyzacji. Siła regularyzacji wzrasta wprost proporcjonalnie do jego wartości. W implementacjach programistycznych współczynnik ten może być zastąpiony parametrem C, który wówczas należy interpretować jako odwrotność λ (Raschka & Mirjalili, 2019, pp. 86-87).

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_{i,k}\, ln\left(\hat{p}_{i,k}\right) + \frac{\lambda}{2}\|\theta\|^2$$

(9)

3.5.2 Liniowy klasyfikator SVM

Celem liniowego klasyfikatora SVM (z ang. *Support Vector Machine*) nazywanego również maszyną wektorów nośnych jest zbudowanie maksymalnie dużego marginesu separującego w przestrzeni punkty należące do dwóch odrębnych klas.

Pierwszym krokiem budowania takiego modelu jest zdefiniowanie hiperpłaszczyzny zwanej również granicą decyzyjną za pomocą prostego równania liniowego (10), gdzie θ symbolizuje wektor współczynników bez wyrazu wolnego θ_0, a x oznacza wektor zmiennych niezależnych.

Z komentarzem [MG6]: Poprawka

$$\theta^T \cdot x + \theta_0 = 0$$

(10)

Wokół niej należy następnie ustalić dwie hiperpłaszczyzny równoległe, oddalone o tą samą odległość. Każda z nich wyznaczy granicę występowania próbek konkretnej klasy, dzięki czemu powstanie wspomniany margines. Zakładając, że etykiety próbek pochodzą ze zbioru $\{-1, 1\}$, gdzie ujemna wartość etykiety oznacza klasę negatywną, margines zostaje opisany wzorami (11).

$$\begin{cases} \theta^T \cdot x_{poz} + \theta_0 = 1 \\ \theta^T \cdot x_{neg} + \theta_0 = -1 \end{cases} \quad (11)$$

Długość wektora w można z łatwością wyznaczyć (12).

$$\|\theta\| = \sqrt{\sum_{j=1}^{n} \theta_j^2} \quad (12)$$

Odejmując od siebie równania układu (11), a następnie normalizując wynik z użyciem długości wektora w (12) powstaje równanie (13), którego lewą stronę można zinterpretować jako odległość między negatywną i pozytywną hiperpłaszczyzną, czyli poszukiwany margines. Algorytm SVM stara się znaleźć maksymalną wartość wyrażenia $\frac{2}{\|\theta\|}$, co jest równoznaczne z wyznaczeniem największego możliwego marginesu.

$$\frac{\theta^T(x_{poz} - x_{neg})}{\|\theta\|} = \frac{2}{\|\theta\|} \quad (13)$$

Podczas optymalizacji, musi on jednak uwzględnić dodatkowy warunek, że każda próbka i powinna być poprawnie klasyfikowana, to znaczy zależnie od opisującej ją etykiety Y_i, powinna leżeć na lub za wyznaczoną hiperpłaszczyzną marginesu (14).

$$\begin{cases} \theta^T \cdot x_i + \theta_0 \geq 1 & jeśli\ Y_i = 1 \\ \theta^T \cdot x_i + \theta_0 \leq -1 & jeśli\ Y_i = -1 \end{cases} \quad (14)$$

Powyższy warunek może być zapisany w krótszej postaci (15).

$$Y_i(\theta^T \cdot x_i + \theta_0) \geq 1 \quad (15)$$

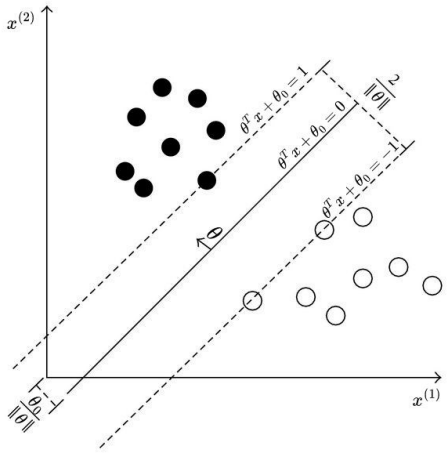
W rzeczywistości zamiast minimalizacji $\|\theta\|$ stosuje się minimalizację wyrażenia $\frac{1}{2}\|\theta\|^2$ ze względu na jego różniczkowalność w punkcie $\theta = 0$ i wygodną w przetwarzaniu pochodną.

Próbki należące do zbioru, których lokalizacja wyznacza położenie marginesu separującego nazywane są wektorami nośnymi (ang. *support vectors*) (Raschka & Mirjalili, 2019, pp. 88-90). Wizualizacja granicy decyzyjnej oraz marginesu separującego próbki dwóch klas została przedstawiona na Rys. 2. Położenie każdego punktu x_i opisane jest przez wartości poszczególnych cech, przy czym na Rys. 2 występują poglądowo dwa takie atrybuty, $x^{(1)}$ oraz $x^{(2)}$.

Z komentarzem [MG7]: Dodano

Rys. 2 – Wyznaczanie hiperpłaszczyzny liniowego klasyfikatora SVM oraz maksymalizacja marginesu separującego. Źródło: platforma ResearchGate[14] z edycją własną

Z komentarzem [MG8]: Można tak dopisać?

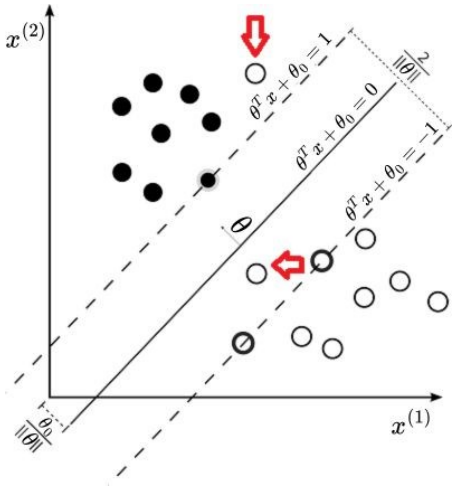


Istnieje modyfikacja algorytmu umożliwiająca rozwiązywanie za pomocą maszyny wektorów nośnych również problemów nieseparowalnych liniowo. Zawiera ona koncepcję tak zwanej zmiennej uzupełniającej (ang. *slack variable*) ξ, z którą nierozerwalnie związana jest idea klasyfikacji miękkiego marginesu (ang. *soft-margin classification*). Dopuszcza ona pewne naruszenia wyznaczonego marginesu, czyli istnienie próbek, które go przekraczają, mimo tego nie definiując nowych jego parametrów. Zjawisko zostało przedstawione na Rys. 3.

[14] https://www.researchgate.net/figure/Linear-SVM-with-maximum-margin-hyperplane_fig6_258524366, dostęp z dnia 09.03.2024 r.

Rys. 3 – Naruszenia miękkiego marginesu. Źródło: platforma StackExchange[15] z edycją własną

Z komentarzem [MG9]: Można tak dopisać?



Przekształcenie algorytmu polega na dodaniu zmiennej ξ do zestawu ograniczeń (14), dzięki czemu do modelu zostaje wprowadzona większa elastyczność (16).

$$\begin{cases} \theta^T \cdot x_i + \theta_0 \geq 1 - \xi_i & jeśli\ Y_i = 1 \\ \theta^T \cdot x_i + \theta_0 \leq -1 + \xi_i & jeśli\ Y_i = -1 \end{cases} \tag{16}$$

W tym przypadku, minimalizuje się wyrażenie nieco inne niż poprzednio, posiadające dodatkowy składnik (17). Parametr C umożliwia kontrolowanie wysokości kary za nieprawidłowe przypisanie do klas. Im większa wartość parametru C, tym bardziej rygorystyczny trening algorytmu. Można uznać tę modyfikację za rodzaj regularyzacji modelu (Raschka & Mirjalili, 2019, pp. 90-92).

$$\frac{1}{2}\|\theta\|^2 + C\left(\sum_i \xi_i\right) \tag{17}$$

Funkcja zawiasowa (ang. *hinge loss*) i jej spotęgowany wariant (ang. *squared hinge loss*) (Rosset, et al., 2004) są najczęściej wybieranymi funkcjami kosztu dla klasyfikatora SVM. Ich wartości zależą od dwóch czynników. Funkcje przypisują większy błąd, gdy występuje różnica w znaku wartości rzeczywistej i przewidywanej. Jeśli znak predykcji jest prawidłowy, natomiast wynik dla obserwacji nie przekracza ustalonego marginesu, przykład również generuje pewien mały koszt[16].

[15] https://stats.stackexchange.com/q/191928, dostęp z dnia 09.03.2024 r.
[16] https://insideaiml.com/blog/Hinge-Loss-and-Square-Hinge-loss-1068, dostęp z dnia 24.03.2024 r.

W przypadku klasyfikacji wieloklasowej za pomocą maszyny wektorów nośnych, najczęściej stosowaną strategią dla tego modelu jest jeden-przeciw-pozostałym (ang. *one-versus-rest*), czyli konstrukcja takiej liczby klasyfikatorów ile jest klas, nie uwzględniając jednej, a następnie analiza uzyskanych wyników dla jednoznacznej predykcji.

Maszyna wektorów nośnych jest klasyfikatorem podobnym do regresji logistycznej, celem obu algorytmów jest optymalna separacja próbek, choć każdy z nich optymalizuje inne kryteria, aby to osiągnąć (Skiena, 2017, pp. 366-369).

3.5.3 Wielomianowy naiwny klasyfikator bayesowski

Naiwne klasyfikatory bayesowskie (ang. *Naive Bayes*) to rodzina klasyfikatorów probabilistycznych. Oznacza to, że przewidują one rozkład prawdopodobieństwa przynależności próbki do zestawu klas, zamiast proponować predykcję do pojedynczej klasy. Ze względu na założenie, że predyktory są parami niezależne, co rzadko jest możliwe w przypadku pracy na realnych danych, nazywa się je naiwnymi. Oparte są na twierdzeniu Bayesa (18).

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)} \tag{18}$$

W powyższym wzorze y oznacza zmienną celu, a x to zależny wektor cech. Zapis $p(y|x)$ określa natomiast prawdopodobieństwo warunkowe, czyli w tym wypadku prawdopodobieństwo wystąpienia y pod warunkiem zajścia x.

Do zadań klasyfikacji tekstu, odpowiednim wariantem z tej rodziny modeli jest wielomianowy naiwny klasyfikator bayesowski. Stara się on parametryzować wielomianowy rozkład danych dla każdej klasy k za pomocą wektorów $\theta_k = (\theta_{k,1}, \dots, \theta_{k,n})$. Oznaczenie n symbolizuje liczbę cech, czyli w wypadku klasyfikacji tekstu – wielkość słownictwa. $\theta_{k,j}$ określa prawdopodobieństwo warunkowe wystąpienia cechy j w próbce należącej do klasy k. Parametry θ_k są szacowane metodą liczenia częstości względnych (19).

$$\hat{\theta}_{k,j} = \frac{N_{k,j} + \alpha}{N_k + \alpha n} \tag{19}$$

W przedstawionym wzorze, $N_{k,j}$ wyraża ilość wystąpień cechy j w próbkach klasy k w zbiorze treningowym, natomiast N_k jest całkowitą liczbą wystąpień wszystkich cech w próbkach klasy k zbioru treningowego[17]. Parametr wygładzania addytywnego został oznaczony literą α.

3.5.4 Drzewo decyzyjne

Model drzewa decyzyjnego (ang. *decision tree*) można opisać jako klasyfikator podejmujący decyzję o etykiecie próbki na podstawie odpowiedzi na szereg pytań opartych na cechach wybranego zbioru treningowego. Budowa takiego drzewa polega na zdefiniowaniu najpierw jego korzenia (ang. *root*), a następnie na iteracyjnym rozdzielaniu danych w węzłach nadrzędnych na węzły potomne według największych wartości przyrostu informacji (ang. *information gain*). Warto ograniczyć parametr maksymalnej głębokości drzewa, aby zapobiec jego zbyt szerokiemu rozrostowi.

Funkcja przyrostu informacji zdefiniowana jest jako różnica między zanieczyszczeniem węzła nadrzędnego a sumą zanieczyszczeń węzłów potomnych (20). Najbardziej pożądanym rezultatem jest podział danych w węźle na jak najbardziej jednolite węzły potomne, wówczas funkcja ta przyjmuje maksymalne wartości.

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{n} \frac{N_j}{N_p} I(D_j) \tag{20}$$

W powyższym wzorze przyjęto oznaczenie I jako wybraną miarę zanieczyszczenia, literę n jako symbol liczby węzłów potomnych, a D_p i D_j jako zestaw danych odpowiednio węzła nadrzędnego i j-tego węzła potomnego. Analogicznie, symbole N_p i N_j określają całkowitą liczbę próbek w tych węzłach. Parametr f definiuje charakterystykę, na podstawie której nastąpi rozgałęzianie. Przedstawione równanie najczęściej sprowadza się jednak do nieco prostszej formy (21), ponieważ ze względu na mniejszą złożoność często stosuje się drzewo binarne, czyli takie, w którym węzeł rozgałęzia się maksymalnie na dwie podgrupy.

$$IG(D_p, f) = I(D_p) - \frac{N_{lewy}}{N_p} I(D_{lewy}) - \frac{N_{prawy}}{N_p} I(D_{prawy}) \tag{21}$$

Istnieje kilka popularnych miar zanieczyszczeń. Jako przykład takiego kryterium może posłużyć wskaźnik Giniego (22). Uzyskuje on najwyższą wartość, gdy rozkład klas w węźle jest

[17] https://scikit-learn.org/stable/modules/naive_bayes.html

równomierny. Wyrażenie $p(k|t)$ symbolizuje odsetek próbek klasy k w węźle t (Raschka & Mirjalili, 2019, pp. 99-101).

$$I_{Gini}(t) = \sum_{k=1}^{K} p(k|t)\big(1 - p(k|t)\big) = 1 - \sum_{k=1}^{K} p(k|t)^2 \tag{22}$$

3.5.5 Ekstremalne wzmocnienie gradientowe

Algorytm ekstremalnego gradientowego wzmocnienia drzew, bardziej znany jako XGBoost, został pierwotnie przedstawiony w 2016 roku (Chen & Guestrin, 2016). Jest to rozszerzenie koncepcji gradientowego wzmacniania drzew charakteryzujące się jednak większą efektywnością i skalowalnością. Warto dodać, że XGBoost operuje na drzewach binarnych.

Wzmocnienie (ang. *boosting*) to strategia uczenia zespołowego, która polega na budowaniu serii słabych modeli, z których każdy uczy się na błędach swojego poprzednika, przez co generuje dokładniejsze od niego predykcje. W ramach szkolenia, seria modeli rozszerza się tak długo o nowe instancje, aż osiągnięty zostanie zdefiniowany wcześniej limit ich liczby lub osiągnięty zostanie warunek stopu. Wzmocnienie gradientowe (ang. *gradient boosting*) jest techniką, w której parametrami uczącymi przekazywanymi do następnika są błędy resztowe poprzednika, zwane też błędami rezydualnymi[18].

Omawiając ideę działania ekstremalnego wzmocnienia gradientowego drzew dla problemu klasyfikacyjnego, najlepiej jest rozpocząć od przedstawienia funkcji celu (23) minimalizowanej podczas budowy danego drzewa.

$$L = \left[\sum_{i=1}^{m} l(Y_i, y_i)\right] + \left[\lambda T + \frac{1}{2}\lambda W_{wyj}^2\right] \tag{23}$$

Powyższe wyrażenie składa się z dwóch części. Pierwsza z nich to funkcja kosztu określona jako suma odstępstw predykcji modelu y_i od rzeczywistej etykiety Y_i każdej z m próbek, po niej natomiast następuje termin regularyzacji z parametrem λ. Symbol T oznacza liczbę końcowych węzłów. Ponieważ składowa λT bierze udział w przycinaniu drzewa kiedy jest ono już w pełni zbudowane, na potrzeby objaśnień można chwilowo ją pominąć. W_{wyj} to wartość wyjściowa rozpatrywanego liścia. Rozpatrując pierwsze drzewo w serii, wyrażenie y_i można

[18] https://www.geeksforgeeks.org/xgboost/, dostęp z dnia 10.03.2024 r.

zastąpić sumą wartości początkowego przypuszczenia y_i^0 i wartości wyjściowej W_{wyj} rozpatrywanego liścia tego drzewa. Poniżej przedstawiono wynikowe równanie (24).

$$L_{uproszcz\_1} = \left[\sum_{i=1}^{m} l\left(Y_i, y_i^0 + W_{wyj}\right)\right] + \left[\frac{1}{2}\lambda W_{wyj}^2\right] \tag{24}$$

Budowa najlepszego możliwego modelu drzewa opiera się na minimalizacji funkcji celu poprzez poszukiwanie optymalnej wartości W_{wyj} dla liści. W kolejnych akapitach przedstawiono etapy rozwiązywania zadanego problemu optymalizacyjnego oraz szereg przekształceń umożliwiających wyrażenie zmiennej W_{wyj} praktyczną w kontekście obliczeń cyfrowych formułą.

Oszacowanie wartości samej funkcji kosztu można przeprowadzić korzystając z rozwinięcia Taylora[19] drugiego rzędu, który jednocześnie sprowadza jej wzór do bardziej elementarnej formy (25). Według nazewnictwa stosowanego w definicji algorytmu XGBoost, pierwsza pochodna funkcji kosztu zostaje zastąpiona oznaczeniem g symbolizującym gradient, a druga pochodna wyrażona jest literą h jako reprezentacja hessiana.

$$l\left(Y_i, y_i + W_{wyj}\right) \approx l(Y_i, y_i) + \left[\frac{d}{dy_i} l(Y_i, y_i)\right] W_{wyj} + \frac{1}{2}\left[\frac{d^2}{dy_i^2} l(Y_i, y_i)\right] W_{wyj}^2 = l(Y_i, y_i) + g_i W_{wyj} + \frac{1}{2} h_i W_{wyj}^2 \tag{25}$$

Teraz można rozwinąć wzór (24) i podstawić wynik przekształcenia (25) w miejsce funkcji kosztu. Zabieg ten został przedstawiony w równaniu (26).

$$\begin{aligned} L_{uproszcz\_2} &= l\left(Y_1, y_1^0 + W_{wyj}\right) + l\left(Y_2, y_2^0 + W_{wyj}\right) + \cdots + l\left(Y_m, y_m^0 + W_{wyj}\right) + \frac{1}{2}\lambda W_{wyj}^2 \\ &= l(Y_1, y_1^0) + g_1 W_{wyj} + \frac{1}{2} h_1 W_{wyj}^2 + l(Y_2, y_2^0) + g_2 W_{wyj} + \frac{1}{2} h_2 W_{wyj}^2 + \cdots + l(Y_m, y_m^0) \\ &+ g_m W_{wyj} + \frac{1}{2} h_m W_{wyj}^2 + \frac{1}{2}\lambda W_{wyj}^2 \end{aligned} \tag{26}$$

Warto zwrócić uwagę na to, że w powyższej formule znajdują się człony nie zawierające elementu W_{wyj}. Przed przeszukiwaniem wartości tego komponentu w celu optymalizacji funkcji celu, można uprościć wzór eliminując wspomniane człony, ponieważ nie mają one wpływu na zmienną W_{wyj}. Następnie należy uporządkować otrzymane wyrażenie (27). Warto podkreślić, że po uproszczeniu wzór (26) nie jest już równoważny formule (27), ponieważ

Z komentarzem [MG10]: Skontrolować numerki czy się nie przesunęły

[19] https://mathspace.pl/matematyka/genialny-wzor-taylora-czyli-o-informacji-zakodowanej-w-pochodnych/, dostęp z dnia 10.03.2024 r.

została z niego wyeliminowana pewna stała. Nie wpływa to jednak na wartość rozwiązania samego problemu optymalizacyjnego.

$$L_{uproszcz\_3} = g_1 W_{wyj} + \frac{1}{2} h_1 W_{wyj}^2 + g_2 W_{wyj} + \frac{1}{2} h_2 W_{wyj}^2 + \ldots + g_m W_{wyj} + \frac{1}{2} h_m W_{wyj}^2 + \frac{1}{2} \lambda W_{wyj}^2$$
$$= (g_1 + g_2 + \cdots + g_m) W_{wyj} + \frac{1}{2} (h_1 + h_2 + \cdots + h_m + \lambda) W_{wyj}^2$$

(27)

Wyłonienie optymalnej wartości W_{wyj} oznacza znalezienie punktów będących rozwiązaniami równania różniczkowego (28).

$$\frac{d}{dW_{wyj}} (g_1 + g_2 + \cdots + g_m) W_{wyj} + \frac{1}{2} (h_1 + h_2 + \cdots + h_m + \lambda) W_{wyj}^2 = 0$$
$$\Leftrightarrow (g_1 + g_2 + \cdots + g_m) + (h_1 + h_2 + \cdots + h_m + \lambda) W_{wyj} = 0$$
$$\Leftrightarrow (h_1 + h_2 + \cdots + h_m + \lambda) W_{wyj} = -(g_1 + g_2 + \cdots + g_m)$$
$$\Leftrightarrow W_{wyj} = \frac{-(g_1 + g_2 + \cdots + g_m)}{(h_1 + h_2 + \cdots + h_m + \lambda)}$$

(28)

Symbole g oraz h należy zastąpić odpowiednikiem gradientu i hessianu funkcji kosztu. Dla przykładu, można przyjąć na jej miejsce logarytmiczną funkcję straty dla problemu binarnego opisaną dla pojedynczego przykładu wzorem (29).

$$l(Y_i, y_i) = -[Y_i \, ln(y_i) + (1 - Y_i) \, ln(1 - y_i)]$$

(29)

Przed wyznaczeniem g i h, należy nałożyć funkcję logitową na prawdopodobieństwa y_i *(30).*

$$l\left(y_i, ln\left(\frac{y_i}{1 - y_i}\right)\right) = -Y_i ln\left(\frac{y_i}{1 - y_i}\right) + ln\left(1 + exp\left(ln\left(\frac{y_i}{1 - y_i}\right)\right)\right)$$

(30)

Gradientem (31) nazywana jest pierwsza pochodna funkcji, hessianem (32) natomiast jej druga pochodna. Po wyznaczeniu pochodnych należy zastosować przekształcenie odwrotne do funkcji logitowej i tym samym wyeliminować wyrażenia zawierające funkcję eksponencjalną oraz logarytmy.

$$g_i = \frac{d}{dln\left(\frac{y_i}{1 - y_i}\right)} l\left(Y_i, ln\left(\frac{y_i}{1 - y_i}\right)\right) = -Y_i + \frac{exp\left(ln\left(\frac{y_i}{1 - y_i}\right)\right)}{1 + exp\left(ln\left(\frac{y_i}{1 - y_i}\right)\right)} = -(Y_i - y_i)$$

(31)

$$h_i = \frac{d^2}{dln^2\left(\frac{y_i}{1-y_i}\right)} l\left(Y_i, ln\left(\frac{y_i}{1-y_i}\right)\right) = \left(\frac{exp\left(ln\left(\frac{y_i}{1-y_i}\right)\right)}{1+exp\left(ln\left(\frac{y_i}{1-y_i}\right)\right)}\right)\left(\frac{1}{1+exp\left(ln\left(\frac{y_i}{1-y_i}\right)\right)}\right) = y_i(1-y_i) \quad (32)$$

Kolejnym krokiem jest podstawienie otrzymanych wyrażeń do ostatecznej formy wzoru na W_{wyj} (28). Oznaczając rezyduum $(Y_i - y_i)$ jako r_i powstaje formuła (33).

$$W_{wyj} = \frac{-(g_1 + g_2 + \cdots + g_m)}{(h_1 + h_2 + \cdots + h_m + \lambda)} = \frac{r_1 + r_2 + \cdots + r_m}{y_1(1-y_1) + y_2(1-y_2) + \cdots + y_m(1-y_m) + \lambda} = \frac{\sum_{i=1}^{m} r_i}{\sum_{i=1}^{m}[y_i(1-y_i)] + \lambda} \quad (33)$$

Omawiając algorytm XGBoost należy również zdefiniować miarę podobieństwa S. Wyraża się ją wzorem podobnym do wartości wyjściowej liścia, jednak licznik ułamka jest podniesiony do drugiej potęgi (34). Dzięki mierze podobieństwa, możliwe jest obliczenie zysku G dla wybranego podziału węzła nadrzędnego na węzły potomne (35). Umożliwia to określenie, która z dostępnych dla danego zbioru reguł jest najskuteczniejsza dla aktualnego stanu rozbudowy modelu.

$$S = \frac{(\sum_{i=1}^{m} r_i)^2}{\sum_{i=1}^{m}[y_i(1-y_i)] + \lambda} \quad (34)$$

$$G = S_{lewy} + S_{prawy} + S_{korzeń} \quad (35)$$

Istnieje jeszcze jeden wskaźnik ściśle powiązany z ekstremalnym gradientowym wzmacnianiem drzew – minimalna dopuszczalna liczba rezyduów w liściu. Przed dołączeniem nowego liścia obliczana jest dla niego wysokość wskaźnika R (36). Następnie algorytm porównuje ją z wartością ustaloną domyślnie. Jeśli wartość domyślna jest wyższa od uzyskanej, liść nie zostanie utworzony.

$$R = \sum_{i=1}^{m}[y_i(1-y_i)] \quad (36)$$

Przycinanie wygenerowanego drzewa odbywa się na podstawie wyniku odejmowania ustalonego parametru γ od wartości G najniższej istniejącej gałęzi. Jeśli opisana różnica jest

ujemna, gałąź zostaje usunięta[20]. Ten proces powtarzany jest iteracyjnie aż do napotkania gałęzi o pozytywnej wartości $G - \gamma$.

Schemat modelowania za pomocą algorytmu XGBoost rozpoczyna się od wyznaczenia wartości początkowego przypuszczenia y_i^0 i obliczenia na tej podstawie błędów rezydualnych r_i dla wszystkich próbek zbioru treningowego. Ponadto, zdefiniowany zostaje parametr regularyzacji λ. Za pomocą miary podobieństwa S oraz wartości zysku G wyznaczane są najlepsze możliwe podziały węzłów, jednak z uwzględnieniem wskaźnika minimalnej dopuszczalnej liczby rezyduów w liściu. W procesie iteracyjnym powstaje pierwsze drzewo binarne, które następnie jest przycinane. Kolejny krok to obliczenie wartości wyjściowych W_{wyj} indywidualnie dla każdego liścia w drzewie. Dla każdej próbki wartość wyjściowa liścia, w którym ona się znajduje, jest mnożona przez parametr uczenia ε, a do uzyskanego iloczynu dodawana jest funkcja logitowa z wartości początkowego przypuszczenia y_i^0. Po konwersji wyników na prawdopodobieństwa z użyciem funkcji logistycznej, można zaktualizować wartości prawdopodobieństw y_i dla poszczególnych próbek. Nowe błędy rezydualne powinny być mniejsze w porównaniu do poprzedników. Następna iteracja polega na budowie nowego drzewa z tą różnicą, że proces trenowania opiera się tym razem na nowych wartościach y_i, które nie są już identyczne dla wszystkich przykładów. Wartości wyjściowe drugiego drzewa są również mnożone przez parametr ε i dodawane do wszystkich poprzednich predykcji. Szereg modeli zostaje zasilony o nowy element. Analogicznie do poprzednika, model definiuje nowe wartości y_i. W ten sposób w algorytmie XGBoost budowane są kolejne drzewa aż do osiągnięcia warunku stopu[21].

3.6 Sztuczne sieci neuronowe

Sztuczna sieć neuronowa (ang. *Artificial Neural Network*) to koncepcja inspirowana działaniem ludzkiego układu nerwowego powstała pierwotnie w 1943 roku (McCulloch & Pitts, 1943). Najmniejszą jednostką sieci jest neuron, który na wzór swojego biologicznego odpowiednika ma za zadanie przesyłanie sygnału, informacji. Posiada on blok sumujący

[20] Opisane w tym podrozdziale szczegóły matematyczne rządzące algorytmem XGBoost są oparte na materiale kanału StatQuest with Josh Starmer pod tytułem *XGBoost Part 3 (of 4): Mathematical Details*. Źródło: https://youtu.be/ZVFeW798-2I?si=PFxl8_h-DmoEmoNx, dostęp z dnia 10.03.2024 r.

[21] Opisany w podrozdziale schemat modelowania algorytmu XGBoost jest oparty na materiale kanału StatQuest with Josh Starmer pod tytułem *XGBoost Part 2 (of 4): Classification*. Źródło: https://youtu.be/8b1JEDvenQU?si=k7-RcAeUeD5JzCba, dostęp z dnia 10.03.2024 r.

sparametryzowany wagami, którego wynik przechodzi przez funkcję aktywacji zwracając następnie jedno wyjście.

Istnieje kilka popularnych funkcji aktywacji, dobrze sprawdzających się w sieciach neuronowych. Najczęściej stosuje się funkcję sigmoidalną, tangens hiperboliczny, ReLU i jej odmiany oraz Softmax. Zadaniem tego bloku jest sprowadzenie wyników obliczeń neuronu do określonego przedziału wartości. Na tej podstawie podejmowana jest decyzja, czy dany sygnał zostanie przesłany dalej, do następnych komórek.

Sztuczna sieć neuronowa złożona jest przynajmniej z trzech warstw neuronów: wejściowej – przyjmującej i wstępnie przetwarzającej dane, minimum jednej warstwy ukrytej, która analizuje otrzymane informacje i wychwytuje charakterystyki próbek oraz warstwy wyjściowej, która dokonuje ostatecznego podsumowania i zwraca pewną decyzję[22]. Wybór architektury sieci jest ściśle zależny od rozwiązywanego problemu badawczego. Sieć zawierająca więcej niż jedną warstwę ukrytą nazywana jest głęboką (ang. *Deep Neural Network*).

Oprócz warstwy gęstej (ang. *dense layer*), której każdy neuron posiada indywidualne połączenie z wszystkimi neuronami poprzedniej warstwy opisane odrębną wagą, dostępnych jest wiele innych rodzajów warstw do budowy sztucznych sieci neuronowych. Stworzona architektura może być złożeniem różnych typów warstw, z których każdy ma do wykonania inne zadanie. W przetwarzaniu sekwencji kluczowe może być ubogacenie struktury sieci komórką rekurencyjną posiadającą pewien rodzaj pamięci do poprzednich elementów przetwarzanego ciągu, a analizując obrazy warto dodać instancję warstwy konwolucyjnej, konsolidującej informacje wychwycone na poziomie pikseli w skoncentrowaną formę. Innym popularnym elementem chętnie stosowanym w sieciach neuronowych jest warstwa łącząca (ang. *pooling layer*) służąca do redukcji wymiarowości map zdefiniowanych na próbkach cech (ang. *feature maps*) i przez to próbująca uchronić model przed przeuczeniem, a także zmniejszająca jego złożoność obliczeniową. Architekturę sieci można również dopełnić segmentami normalizującymi dane na poziomie poszczególnych obserwacji (ang. *layer normalization*) (Ba, et al., 2016) lub paczek danych (ang. *batch normalization*) (Ioffe & Szegedy, 2015) w celu stabilizacji treningu. Z kolei blok porzucania (ang. *dropout*) jest jedną z najpopularniejszych technik regularyzacji sieci neuronowej obok regularyzacji L2 dążącej do

[22] https://www.sztucznainteligencja.org.pl/kurs/sztuczna-inteligencja-dla-poczatkujacych/sztuczne-sieci-neuronowe/, dostęp z dnia 12.03.2024 r.

zmniejszenia wag modelu. Polega ona na losowym pomijaniu niektórych neuronów nienależących do warstwy wyjściowej w poszczególnych przebiegach treningu. Dzięki temu zapobiega przeuczeniu modelu i skraca czasu treningu. Celem technik regularyzacji jest poprawienie generalizacji sieci.

Uczenie sieci neuronowej opiera się na zastosowaniu algorytmu propagacji wstecznej (ang. *backpropagation*). Mechanizm ten został uznany za przełomowy w rozwoju sztucznej inteligencji, ponieważ zawierał odpowiedź na pytanie, jak trenować sztuczną sieć neuronową w sposób skuteczny i wydajny (Rumelhart, et al., 1986). Algorytm polega na przeprowadzaniu kolejnych próbek treningowych lub paczek danych (ang. *batches*) przez sieć, generując wstępną predykcję, następnie obliczeniem popełnianego przez nią błędu porównując otrzymaną predykcję z wartościami wyniku oczekiwanego. Kolejnym krokiem jest mierzenie wpływu neuronów w kolejnych warstwach na błędny wynik predykcji zaczynając od warstwy wyjściowej w kierunku warstwy wejściowej i na tej podstawie modyfikacja parametrów poszczególnych neuronów. Fundamentem techniki propagacji wstecznej jest metoda gradientu prostego (Géron, 2018, pp. 262-263).

Warto dodać, że w przypadku złożonych problemów i stosowania głębokich sieci modele są narażone na wystąpienie problemu zanikających lub eksplodujących gradientów (Bengio, et al., 1994), dodatkowo stają się kosztowne obliczeniowo, a także narażone na przetrenowanie. Oprócz metod bazujących na modyfikacji parametrów i struktury modelu, na przykład zmiany strategii inicjowania wag połączeń lub typów funkcji aktywacji czy dodania segmentów regularyzujących, nieocenionym rozwiązaniem może być zastosowanie optymalizatorów, np. algorytmu Adam (ang. *Adaptive Moment Estimation*) (Kingma & Ba, 2014), które są szybsze od tradycyjnej techniki gradientu prostego (Géron, 2018, pp. 275-302).

3.6.1 Rekurencyjne sieci neuronowe

Sieci rekurencyjne (ang. *Recurrent Neural Networks*) są odpowiednim wyborem dla danych sekwencyjnych, czyli takich, w których kolejność poszczególnych elementów próbki ma znaczenie. W tym przypadku, znaczące w modelowaniu jest uwzględnienie relacji między następującymi po sobie fragmentami ciągów. W tym celu powstała koncepcja neuronu rekurencyjnego. W każdym takcie t, zwanym także krokiem czasowym (ang. *time step*) taki neuron oprócz danych wejściowych x_t otrzymuje również własne wyniki z poprzedniego taktu

y_{t-1}, przez co analizuje dany element sekwencji uwzględniając informacje pozyskane od jego poprzedników.

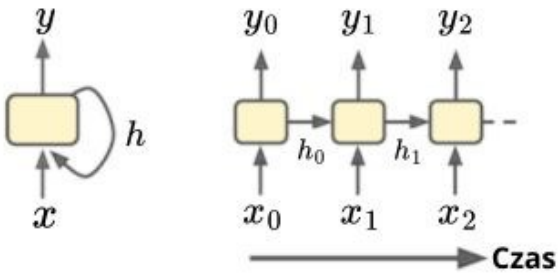
Z neuronów rekurencyjnych można utworzyć prostą warstwę. Wówczas, analogicznie do działania pojedynczej jednostki, każdy z neuronów otrzymuje w takcie t dane wejściowe x_t oraz swój poprzedni wynik y_{t-1}, jednak tym razem różnica polega na wymiarowości – wejście i wyjście z warstwy są w postaci wektorów. Każdy neuron posiada dwa rodzaje wag. Wagi wszystkich neuronów dla danych wejściowych zawierają się w macierzy W_x, a dla danych wyjściowych z poprzedniego taktu znajdują się w tablicy W_y. Wyniki opisanej warstwy dla pojedynczej próbki można wyrazić wzorem (37), gdzie ϕ oznacza funkcję aktywacji, najczęściej ReLU lub tangens hiperboliczny, a b to obciążenia każdego neuronu.

$$y_t = \phi\left(W_x^T \cdot x_t + W_y^T \cdot y_{t-1} + b\right) \tag{37}$$

Rozważając istotę przedstawionych struktur, można wywnioskować, że zawierają one coś na kształt pamięci, ponieważ wyjście w danym kroku czasowym t jest funkcją wszystkich danych wejściowych z poprzednich taktów. Dlatego w profesjonalnym nazewnictwie funkcjonują terminy komórka (ang. *cell*) i komórka pamięci (ang. *memory cell*) określające neuron rekurencyjny, warstwę rekurencyjną, a także inne, bardziej zaawansowane odmiany tych mechanizmów. Stan ukryty komórki h_t opisywany jest wzorem $h_t = f(h_{t-1}x_t)$. Pojęcie to zostało wyodrębnione, ponieważ w bardziej skomplikowanych komórkach wyjście y_t nie będzie równoważne z wartością jej stanu h_t. Działanie komórki rekurencyjnej na przestrzeni czasu nazywa się powszechnie rozwijaniem w czasie (ang. *unrolling through time*). Proces ten został przedstawiony na Rys. 4 obok schematu komórki rekurencyjnej z oznaczeniem jej stanu ukrytego.

Rys. 4 – Komórka rekurencyjna i jej stan ukryty (po lewej) oraz jej rozwijanie w czasie (po prawej). Źródło: (Géron, 2017) z edycją własną

Z komentarzem [MG11]: Można tak dopisać?



Trening sieci rekurencyjnej polega na jej rozwinięciu w czasie, a następnie na implementacji metody wstecznej propagacji błędu w czasie. Po inferencji próbek, funkcja kosztu ocenia trafność nieignorowanych wyjść sieci na podstawie realnych etykiet zbioru treningowego. Informacja o gradientach tej funkcji jest posyłana wstecz poprzez uwzględniane wyjścia, a następnie rozprowadzone po całej rozwiniętej sieci, czyli z uwzględnieniem więcej niż ostatniego taktu.

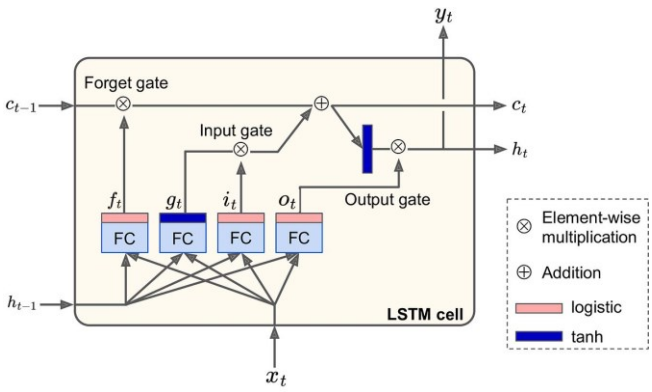
Istnieją różne architektury sieci rekurencyjnych. W zależności od problemu badawczego i pożądanych rezultatów można inaczej zdefiniować liczbę neuronów w poszczególnych komórkach, ustalić, które wyjścia kroków czasowych są istotne, a które ignorowane, zdefiniować, czy w każdym takcie podawany będzie kolejny element sekwencji, czy jedynie w pierwszym. Do segmentu rekurencyjnego można dołączyć inne rodzaje warstw specyficznych dla rozwiązania problemu, na przykład komponent Softmax przy zadaniu klasyfikacji sekwencji.

Uczenie standardowej sieci rekurencyjnej na długich sekwencjach danych może skutkować pojawieniem się problemu zanikającego lub eksplodującego gradientu, a ponadto wymagać wiele czasu. Dodatkowo, początek długiego ciągu elementów może być przez taki model zapomniany wraz z postępowaniem analizy naprzód i kolejnymi taktami. W odpowiedzi na zaistniałe potrzeby, w 1997 roku zaproponowano alternatywną strukturę w postaci komórki LSTM (ang. *LSTM cell*) (Hochreiter & Schmidhuber, 1997) będącej pewnym wariantem tradycyjnej komórki rekurencyjnej. Innowacyjne rozwiązanie wprowadziło do komórki element pamięci długotrwałej i zyskało popularność stając się podstawą wielu funkcjonalnych projektów uczenia maszynowego.

Najbardziej charakterystyczną cechą opisywanego bloku jest pamięć długoterminowa c_t występująca zaraz obok stanu ukrytego h_t, który działa na wzór pamięci krótkoterminowej. Na Rys. 5 przedstawiono szczegółową budowę komórki LSTM. Nadrzędnym celem poszczególnych części mechanizmu jest umiejętne operowanie informacjami i zarządzanie pamięcią, czyli skuteczne definiowanie, które wiadomości zawarte w próbkach należy zachować i umożliwić efektywne wracanie do nich, a które eliminować i przez to zwalniać miejsce dla nowych danych.

Rys. 5 – Struktura komórki LSTM. Źródło: (Géron, 2017) z edycją własną

Z komentarzem [MG12]: Można tak dopisać?



W komórce LSTM wektor wejściowy x_t i poprzedni wektor stanu h_{t-1} są przekazywane do czterech różnych warstw. Główna warstwa występuje również w podstawowej komórce rekurencyjnej i zwraca wartość g_t po przetworzeniu uzyskanych danych. Bramka zapominająca (ang. *forget gate*) decyduje, które informacje w pamięci długotrwałej zostaną usunięte, jej wyjście stanowi wektor f_t. Bramka wejściowa (ang. *input gate*) przyczynia się do częściowego zachowania informacji zwracanych przez warstwę główną poprzez selekcję elementów wyjścia g_t, które należy dodać do pamięci długoterminowej. Jej wektor wynikowy oznaczany jest symbolem i_t. Bramka wyjściowa (ang. *output gate*) definiuje składniki stanu długoterminowego, które są brane pod uwagę przy generowaniu nowych wektorów h_t i y_t w danym takcie t. Jej wyjście opisuje się jako o_t. Warto dodać, że warstwa główna jako jedyna używa tangensa hiperbolicznego w roli funkcji aktywacji. Pozostałe trzy bramki korzystają z

funkcji logistycznej. Równania (38), (39), (40) i (41) opisują operacje wykonywane przez poszczególne warstwy w kroku obliczeniowym t dla pojedynczej próbki.

$$i_t = \sigma(W_{xi}^T \cdot x_t + W_{hi}^T \cdot h_{t-1} + b_i) \tag{38}$$

$$f_t = \sigma(W_{xf}^T \cdot x_t + W_{hf}^T \cdot h_{t-1} + b_f) \tag{39}$$

$$o_t = \sigma(W_{xo}^T \cdot x_t + W_{ho}^T \cdot h_{t-1} + b_o) \tag{40}$$

$$g_t = tanh(W_{xg}^T \cdot x_t + W_{hg}^T \cdot h_{t-1} + b_g) \tag{41}$$

Wygenerowanie nowego stanu pamięci długoterminowej c_t polega na przepuszczeniu poprzedniego stanu c_{t-1} przez bramkę zapominającą, gdzie usuwana jest część jego zawartości, a następnie na zsumowaniu (ang. *addition*) wynikowego wektora z nowymi znaczącymi informacjami wybranymi przez bramkę wejściową. Zdefiniowanie wartości h_t, polega natomiast na obliczeniu tangensa hiperbolicznego dla kopii otrzymanego stanu c_t, a rezultat tego działania jest filtrowany za sprawą pomnożenia (ang. *multiplication*) przez wytyczne wygenerowane przez bramkę wyjściową. Wartość h_t w tym przypadku jest równa y_t. Obliczenia stanów pamięci dla pojedynczej próbki zostały opisane równaniami (42) i (43).

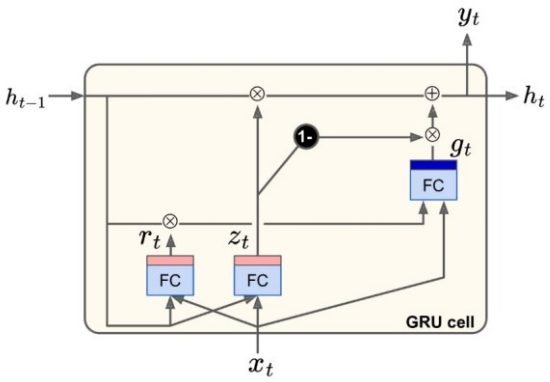
$$c_t = f_t \otimes c_{t-1} + i_t \otimes g_t \tag{42}$$

$$h_t = y_t = o_t \otimes tanh(c_t) \tag{43}$$

W 2014 roku opublikowano uproszczony wariant komórki LSTM osiągający jednak podobne rezultaty. Jednostka GRU (ang. *GRU cell*) (Cho, et al., 2014) odznacza się wprowadzeniem strategii współdziałania bramki zapominającej i wejściowej, a także używaniem tylko jednego stanu pamięci h_t, który jest w każdym takcie zwracany przez komórkę w całości. Jest to rezultat między innymi wyeliminowania bramki wyjściowej na rzecz segmentu kontrolującego dane pojawiające się w warstwie głównej. Strukturę komórki GRU obrazuje Rys. 6.

Rys. 6 – Struktura komórki GRU. Źródło: (Géron, 2017) z edycją własną

Z komentarzem [MG13]: Można tak dopisać?



Poniżej przedstawiono wzory opisujące działanie poszczególnych warstw dla pojedynczego przykładu w takcie t (44)(45)(46). Formuła (47) określa w jaki sposób uzyskuje się wartości nowego stanu pamięci (Géron, 2018, pp. 377-401).

$$z_t = \sigma(W_{xz}^T \cdot x_t + W_{hz}^T \cdot h_{t-1} + b_z) \tag{44}$$

$$r_t = \sigma(W_{xr}^T \cdot x_t + W_{hr}^T \cdot h_{t-1} + b_r) \tag{45}$$

$$g_t = tanh(W_{xg}^T \cdot x_t + W_{hg}^T \cdot (r_t \otimes h_{t-1}) + b_g) \tag{46}$$

$$h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes g_t \tag{47}$$

3.7 Mechanizm Attention

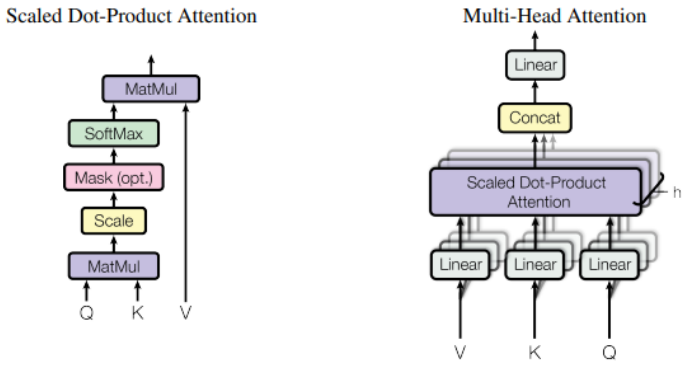
W 2017 roku, w szeroko później cytowanym artykule pod tytułem „Attention Is All You Need" (Vaswani, et al., 2023), w dosłownym tłumaczeniu „Attention to wszystko, czego potrzebujesz" autorzy przedstawili propozycję nowej przełomowej architektury sieci neuronowej typu transformer, której skuteczność oparta była przede wszystkim na istniejącym wówczas lecz niezbyt spopularyzowanym mechanizmie Attention.

Jak wskazuje nazwa, celem algorytmu jest znalezienie i podkreślenie cech próbki, na których należy się skupić podczas modelowania, poprzez przypisanie im wyższych wag. Spośród reprezentacji poszczególnych słów w próbce (ang. *keys*), mechanizm wychwytuje te wysoko skorelowane z wybranym zapytaniem (ang. *query*) kalkulując iloczyn skalarny tablic.

Następnie aktywuje funkcję Softmax, dzięki której poszczególne prawdopodobieństwa sumują się do 1. Dzięki porównaniu z wagami opisującymi cechy znaczące dla rozwiązywanego zadania (ang. *value*) skutkuje wzmocnieniem wag dla istotnych komponentów tekstu i osłabieniem wag dla mniej ważnych. Mechanizm nazywany jest Self-Attention, jeżeli obsługuje różne części tej samej sekwencji wejściowej[23].

Autorzy wspomnieli również o drugim wariancie mechanizmu. Dopuszcza on równoległe działanie kilku modułów obliczeniowych, tak zwanych głów (ang. *heads*). Takie podejście umożliwia równoczesne analizowanie kilku kontekstów, dzięki czemu możliwe jest jednoczesne wychwycenie kilku znaczeń danego słowa. Schematy obu architektur przedstawiono na Rys. 7.

Rys. 7 – Wizualizacja architektur mechanizmu Attention. Źródło: (Vaswani, et al., 2023)



3.8 Trening i ewaluacja modeli

3.8.1 Walidacja krzyżowa

Zastosowanie walidacji krzyżowej ma na celu ewaluację wybranej architektury modelu uczenia nadzorowanego poprzez przeprowadzenie kilkukrotnego szkolenia i testu, za każdym razem na podstawie innych zestawów próbek. Najpopularniejszym wariantem tej metody jest walidacja k-krotna, która polega na rozbiciu zbioru danych na k podzbiorów, a następnie

[23] https://medium.com/@wwydmanski/whats-the-difference-between-self-attention-and-attention-in-transformer-architecture-3780404382f3, dostęp z dnia 30.03.2024 r.

przeprowadzeniu k treningów wybranej architektury, za każdym razem traktując jako zbiór testowy inną spośród stworzonych grup. Ogólną jakość modelu można ocenić uśredniając wyniki k takich klasyfikacji. Jako kryterium oceny skuteczności modelu zazwyczaj przyjmuje się miarę dokładności.

3.8.2 Sprawdzian prosty

Jedno z najczęściej stosowanych podejść do trenowania modeli uczenia maszynowego polega na podziale danych na podzbiory treningowy i testowy, gdzie pojemność zbioru testowego wynosi zazwyczaj 30% wszystkich próbek lub mniej. Wykonywany jest wówczas tylko jeden trening modelu na zbiorze treningowym, natomiast ewaluacja odbywa się poprzez inferencję zbioru testowego, porównanie otrzymanych predykcji z realnymi etykietami i kalkulację pożądanych metryk.

3.8.3 Ewaluacja modelu

Dokładność (ang. *accuracy*) jest jedną z prostych metryk oceny skuteczności modelu klasyfikacyjnego. Jest to stosunek ilości poprawnie zaklasyfikowanych próbek do ilości wszystkich próbek. Należy pamiętać, że ta miara może nie być wystarczająca w przypadku zbioru o nierównomiernym rozkładzie danych na przestrzeni klas, ponieważ mimo otrzymania wysokiej wartości pomiaru, istnieje ryzyko, że model nie rozpoznaje prawidłowo żadnej próbki należącej do klasy rzadkiej.

W procesie ewaluacji modelu warto dodatkowo zmierzyć wskaźnik F_1. Jest to kombinacja dwóch innych metryk: precyzji (ang. *precision*) i czułości (ang. *recall*). Opisuje się go podręcznikową formułą (48).

$$F_1 = 2 * \frac{precyzja * czułość}{precyzja + czułość} \tag{48}$$

Precyzję można traktować jako miarę jakości. Koncentruje się wyłącznie na poprawności przewidywań pozytywnych, nie skupiając się na poprawnym wykrywaniu przewidywań negatywnych. Wyznacza odsetek poprawnie przewidzianych etykiet wśród wszystkich pozytywnych predykcji zwróconych przez model. Jej wysoki wynik minimalizuje szansę na wystąpienie zdarzeń błędnie zidentyfikowanych jako pozytywne, czyli tak zwanych „fałszywych alarmów”. Czułość wskazuje natomiast ułamek poprawnie przewidzianych przez

model próbek spośród tych o rzeczywistej pozytywnej etykiecie. Maksymalizacja tej wartości zmniejsza prawdopodobieństwo niewykrycia istniejącego zdarzenia. Wysoka wartość wskaźnika F_1 oznacza, że model osiągnął balans między tymi dwoma miarami[24].

[24] https://encord.com/blog/f1-score-in-machine-learning/, dostęp z dnia 24.03.2024 r.

4 Materiały i metody

4.1 Wstępne przetwarzanie tekstu

Rozważając problem klasyfikacji próbek tekstu w postaci postów internetowych do kilkunastu kategorii należy najpierw zdefiniować, jakie cechy wpisów mogą okazać się najbardziej istotne dla ich rozróżnienia między sobą i te właśnie atrybuty należy następnie wyeksponować. Jest to istotne, ponieważ ostatecznym celem projektu jest znalezienie indywidualnych schematów obecnych w postach osób będących w obrębie tego samego typu osobowości. Z perspektywy treningu modelu, ważne jest natomiast ograniczenie ilości mniej ważnych słów w bazie danych, ponieważ dzięki temu uczenie trwa krócej przy zachowaniu tej samej ilości informacji.

Początkowo, zmapowano typy osobowości wyrażone literami do wartości liczbowych z zakresu od 0 do 15 i dołączono dodatkową kolumnę z tymi wartościami do ramki danych powstałej po wczytaniu zbioru *(MBTI) Myers-Briggs Personality Type Dataset*. Poszczególne posty danego użytkownika, oddzielone wstępnie symbolami „|||" zostały złączone ze sobą i były później traktowane jako pojedyncza próbka.

Wszystkie litery zostały zmniejszone. W celu tokenizacji przetestowano dwa rozwiązania języka Python: standardowy *word_tokenize*[25] oraz *TweetTokenizer*[26], oba pochodzące z zestawu narzędzi języka naturalnego *NLTK*. W Tabeli 2 przedstawiono porównanie ich działania dla przykładowych fragmentów tekstu. *TweetTokenizer* okazał się być lepiej przystosowany do pracy z postami internetowymi, ze względu na większą czułość na obecność emotikon w tekście, co w praktyce oznacza, że rzadziej rozdziela te charakterystyczne zestawienia znaków. Częściej niż *word_tokenize* odseparowuje on natomiast złączenia losowych symboli z wyrazami („'Now"), co może mieć znaczenie w przypadku błędów w postach pisanych naprędce na forum. Można też zauważyć, że traktuje on linki jako spójną całość, co jest jego niewątpliwą zaletą. *TweetTokenizer* nie rozdziela również angielskich form skróconych („I'm", 'it's'), co jest cechą neutralną – należy zastanowić się, jaka informacja na temat próbek może być decydująca, a jej wydobycie może przełożyć się na większą skuteczność późniejszego modelowania – separacja takich złączeń, pozostawienie ich

[25] https://www.nltk.org/api/nltk.tokenize.word_tokenize.html
[26] https://tedboy.github.io/nlps/generated/generated/nltk.tokenize.TweetTokenizer.html

w pierwotnej formie, a może nawet rozwinięcie wspomnianych skrótów. W każdym z tych podejść można odnaleźć inne charakterystyki – można, na przykład, analizować częstość użycia słów wskazujących na siebie przez poszczególne osoby, wnioskować o temperamencie lub stylu życia ludzi na postawie ich skłonności do używania krótszych form, albo skupić się na badaniu samej treści wypowiedzi po ujednoliceniu języka.

Tabela 2 – Porównanie tokenizacji tekstu przez word_tokenizer i TweetTokenizer. Źródło: opracowanie własne

Fragment próbki 1
‘Now I'm interested. But too lazy to go research it, because it's time-consuming :(
Tokeny zwrócone przez *word_tokenizer* po zmniejszeniu liter
"'now", 'i', "'m", 'interested', '.', 'but', 'too', 'lazy', 'to', 'go', 'research', 'it', ',', 'because', 'it', "'s", 'time-consuming', ':', '('
Tokeny zwrócone przez *TweetTokenizer* po zmniejszeniu liter
"'", 'now', "i'm", 'interested', '.', 'but', 'too', 'lazy', 'to', 'go', 'research', 'it', ',', 'because', "it's", 'time-consuming', ':('
Fragment próbki 2
http://www.youtube.com/watch?v=u8ejam5DP3E On repeat for most of today.
Tokeny zwrócone przez *word_tokenizer* po zmniejszeniu liter
'//www.youtube.com/watch', '?', 'v=u8ejam5dp3e', 'on', 'repeat', 'for', 'most', 'of', 'today', '.'
Tokeny zwrócone przez *TweetTokenizer* po zmniejszeniu liter
'http://www.youtube.com/watch?v=u8ejam5dp3e', 'on', 'repeat', 'for', 'most', 'of', 'today', '.'

W przypadku opisywanej pracy angielskie formy skrócone zostały rozdzielone względem apostrofu, uznane za stosunkowo mało istotne dla badanego problemu klasyfikacji i zupełnie usunięte z próbek wraz z pozostałymi słowami, które same w sobie nic nie znaczą (ang. *stopwords*). Cały proces wstępnego przetworzenia próbki został zwizualizowany w Tabeli 3 na przykładowym fragmencie posta.

Tabela 3 – Proces wstępnej obróbki tekstu. Źródło: opracowanie własne

Fragment próbki
'You're fired. That's another silly misconception.
Zmniejszenie liter i tokenizacja z użyciem *TweetTokenizer*
"'", "you're", 'fired', '.', "that's", 'another', 'silly', 'misconception', '.'
Rozdzielenie form skróconych względem apostrofu
'', '', 'you', 're', 'fired', '.', 'that', 's', 'another', 'silly', 'misconception', '.'
Usunięcie słów bez znaczenia
'', '', 'fired', '.', 'another', 'silly', 'misconception', '.'

4.2 Dodatkowe charakterystyki numeryczne

Za pomocą operacji na wyrażeniach regularnych (ang. *RegEx*), w próbkach zostały zidentyfikowane i oflagowane linki (‘[LINK]’), emotikony min nie zawierające liczb oraz emotikony w kształcie serc, ze względu na ich częste użytkowanie w popkulturze jako symbole pozytywnych emocji (‘[EMOJI]’). Wśród znaków interpunkcyjnych wyróżniono wielokropek, znak zapytania, znak wykrzyknienia oraz kropkę ("[ELLIPSIS]", "[QUESTION_MARK]", "[EXCLAMATION_MARK]", "[PERIOD]"), jako te niosące potencjalnie największą informację. Inne symbole zostały usunięte ze zbioru. Oznaczono również samodzielne liczby (‘[NUMBER]’) oraz usunięto hashtagi, czyli słowa poprzedzone znakiem *#*, funkcjonujące jako słowa kluczowe podczas wyszukiwania i grupowania treści[27]. Na koniec, listy tokenów zostały oczyszczone z pustych elementów powstałych w trakcie procesu. W Tabeli 4 przedstawiono przykłady fragmentów próbek przed i po wykonaniu całego opisanego procesu.

Tabela 4 – Przykłady oznaczania charakterystycznych elementów w próbkach. Źródło: opracowanie własne

Przed	'', '', 'http://www.youtube.com/watch?v=qsxhcwe3krw', 'http://41.media.tumblr.com/tumblr_lfouy03pma1qa1rooo1_500.jpg', 'enfp', 'intj', 'moments'
Po	'[LINK]', '[LINK]', 'enfp', 'intj', 'moments'
Przed	'', '', ',', 'draw', 'nails', '(', 'haha', ')', '.', 'done', 'professionals', 'nails', '.', 'yes', ',', 'gel', '.', 'mean', 'posted', 'done', 'nails', '?', 'awesome', '!'

[27] https://sjp.pl/hashtag, dostęp z dnia 18.11.2023 r.

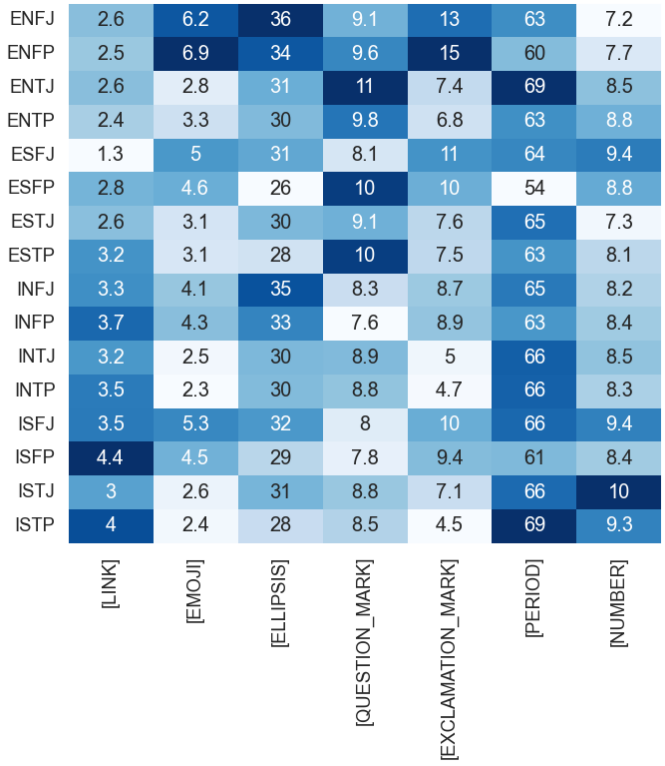
Po	'draw', 'nails', 'haha', '[PERIOD]', 'done', 'professionals', 'nails', '[PERIOD]', 'yes', 'gel', '[PERIOD]', 'mean', 'posted', 'done', 'nails', '[QUESTION_MARK]', 'awesome', '[EXCLAMATION_MARK]'
Przed	'', '', 'always', 'thought', 'tony', 'stark', 'entj', '...', '('
Po	'always', 'thought', 'tony', 'stark', 'entj', '[ELLIPSIS]'
Przed	'', '', 'got', '593', '.', 'read', 'enneagram', '953', ',', 'though', '.'
Po	'got', '[NUMBER]', '[PERIOD]', 'read', 'enneagram', '[NUMBER]', 'though', '[PERIOD]'
Przed	'', '', 'thanks', 'jaydubs', 'miss', 'bingley', 'littledreamer', 'advices', 'important', '!', ':d'
Po	'thanks', 'jaydubs', 'miss', 'bingley', 'littledreamer', 'advices', 'important', '[EXCLAMATION_MARK]', '[EMOJI]'

W celu orientacyjnego sprawdzenia, czy zbiór jest opisywalny garstką stworzonych dodatkowych cech, przekazano próbki wyrażone zawieranymi przez nie licznościami linków, emotikon, liczb oraz poszczególnych znaków interpunkcyjnych do binarnego drzewa decyzyjnego[28], na którym następnie zastosowano walidację krzyżową. Wraz ze wzrostem głębokości drzewa, średnia dokładność malała. Z tego powodu określono maksymalną dopuszczalną głębokość drzewa decyzyjnego wartością 5. Uzyskany uśredniony wynik walidacji krzyżowej dla 10 podzbiorów wyniósł 22.84%, i jest on znacznie wyższy od poziomu losowego, który wynosi 6.25%. To potwierdza, że wyodrębnione cechy zawierają istotne informacje na temat poszczególnych typów osobowości, choć nie definiują ich jednoznacznie.

Z Rys. 8 można wyczytać ile średnio poszczególnych elementów oflagowanych przypada na jednego użytkownika opisanego danym typem osobowości w analizowanym zbiorze danych. Kolumny opisane są nazwami flag, a wiersze czteroliterowymi typami osobowości. Kolory należy porównywać kolumnami – najciemniejsze pole w pionowym zestawieniu oznacza, że użytkownicy o odpowiadającym typie osobowości użyli w swoich postach najwięcej elementów oznaczonych wybraną flagą, a pole o najmniej intensywnym odcieniu określa typ osobowości, którego przedstawiciele dodali ich najmniej. Dodatkowo, pierwsza połowa wierszy dotyczy osób ekstrawertycznych, a druga – introwertycznych.

[28] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Rys. 8 – Liczba oflagowanych elementów przypadająca na jednego użytkownika danego typu osobowości. Źródło: opracowanie własne



Analizując zestawienie, można wywnioskować, że linkami chętniej posługiwały się w swoich wpisach osoby introwertyczne. Dużą ilością wstawianych emotikon wyróżniały się typy ENFP i ENFJ o wspólnych symbolach z aż trzech par kategorii – osoby ekstrawertyczne, kreatywne i stawiające serce ponad rozum, o empatycznym podejściu. Można również zauważyć, że cztery najniższe wyniki związane z zastosowaniem wielokropków należą do wszystkich istniejących czterech typów osobowości ze składowymi „S" oraz „P", czyli osób dających się opisać jako zorientowane na fakty i praktykę oraz lubiące spontaniczny styl życia i pracy. Natomiast najczęściej wielokropków używały wszystkie typy o składowych „N" i „F", czyli zorientowane na możliwości i inspirację, a także podejmujące decyzje sercem i priorytetyzujące harmonię między ludźmi. Można również zaobserwować, że ekstrowertycy

częściej używają pytajników. Ciekawym pomysłem na rozwinięcie tego wątku w przyszłych badaniach byłoby sprawdzenie, czy wynika to z ilości zadawanych przez nich pytań, czy może raczej z zamiłowania do częstego stawiania na końcu zdań serii takich znaków zamiast jednego. W przypadku wykrzykników, tak jak emotikon, ENFJ i ENFP są opisywane wyraźnie najwyższymi wartościami. Kropki używane były dość równomiernie na przestrzeni klas, jednak ich frekwencja w postach typu ESFP jest znacznie niższa. Użytkownika o takiej osobowości można opisać jako towarzyskiego, rzeczowego, empatycznego i spontanicznego. Liczbami najchętniej posługiwały się osoby scharakteryzowane jako ISTJ, czyli introwertyczne i powściągliwe, zorientowane na fakty, kierujące się rozumem i lubiące działać w sztywnych ramach, według opracowanego planu. Dodatkowo, warto dodać, że spośród wszystkich oflagowanych elementów najpowszechniej używane były kropki oraz wielokropki.

4.3 Podział danych

Trening nadzorowanych modeli uczenia maszynowego należy przeprowadzić na pewnym wycinku danych, podczas gdy pozostała część próbek jest wówczas używana do ewaluacji jego skuteczności, a przy tym oceny optymalności wytrenowanych parametrów.

W wypadku tego projektu, zbiór danych został rozbity na podgrupę treningową i testową w stosunku 80:20. Podział nastąpił w sposób stratyfikowany (ang. *stratified split*) na podstawie zmiennej celu, co oznacza zachowanie oryginalnych proporcji pomiędzy klasami decyzyjnymi w każdej z podgrup. W tym celu zastosowano funkcję *train_test_split* biblioteki *scikit-learn*[29]. Podzbiór treningowy wyniósł ponad 6900 próbek, natomiast testowy zawierał powyżej 1700 wierszy.

Posty należące do jednego użytkownika zostały uprzednio scalone w jeden wiersz, dlatego potencjalny problem znalezienia się wpisów tej samej osoby zarówno w zbiorze treningowym jak i testowym nie wystąpił. Taka sytuacja poddałaby w wątpliwość wyniki późniejszej ewaluacji i umiejętności przystosowania się modelu do zupełnie nowych danych.

[29] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

4.4 Klasyfikacja za pomocą tradycyjnych modeli uczenia maszynowego

4.4.1 Przygotowanie reprezentacji wektorowej

Proces konwersji danych tekstowych do postaci wektorowej w przypadku tradycyjnych modeli klasyfikacyjnych obejmował zastosowanie konceptu ważenia częstością termów, czyli rozszerzonej metody worka słów.

W tym celu najpierw zastosowano technikę worka słów z modułu *sklearn.feature_extraction.text*, czyli narzędzie *CountVectorizer*[30]. Parametr *max_features* definiujący rozważaną wielkość słownika ustawiono na wartość 5000. Wynik tej modyfikacji stanowił wejście do funkcji *TfidfTransformer*[31] będącej cyfrową reprezentacją modyfikacji TF-IDF z normalizacją domyślnie wykonywaną za pomocą normy euklidesowej. Oba obiekty zostały dopasowane do danych treningowych metodą *fit_transform*. Reprezentacja danych testowych została natomiast zbudowana przy użyciu metody *transform* na bazie informacji ze zbioru treningowego.

Podczas budowania worka słów najbardziej naturalnym podejściem jest przyjmowanie jako samodzielny element słownika każdego pojedynczego słowa. Istnieje jednak również możliwość traktowania jako jeden człon tak zwanych bigramów, czyli par wyrazów. Jest to uzasadnione podejście, ponieważ w języku naturalnym istnieją słowa, które często występują w parze. Dlatego w tej pracy przeprowadzono trzy różne próby treningu tradycyjnych architektur. Pierwsza z nich bazowała na worku słów zbudowanym z indywidualnych wyrazów, w drugiej słownik zawierał zarówno jedno- jak i dwuczłonowe elementy, trzecia próba opierała się jedynie na bigramach. Przykładowe rekordy poszczególnych worków słów przedstawiono w Tabeli 5. Należy dodać, że eksperymenty dotyczące znajdowania najlepszych parametrów modeli były wykonywane na słowniku zawierającym jedynie pojedyncze wyrazy.

[30] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
[31] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

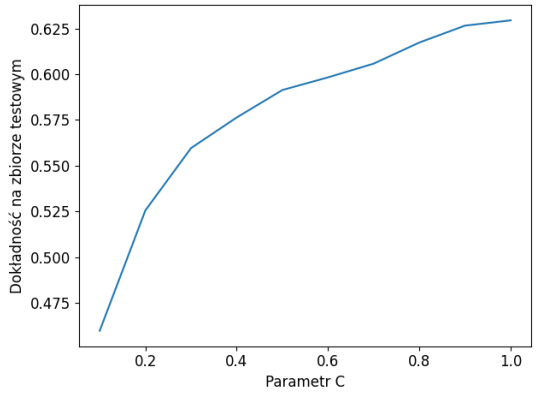
Tabela 5 – Przykładowe rekordy słownika w zależności od numeru próby. Źródło: opracowanie własne

Pierwsza próba	Druga próba	Trzecia próba
care	wings	also tend
others	sorta	new people
tattoos	pointing	star wars
would	gosh	guess ellipsis
never	label	really mean
40	would never	typing people
fox	never get	find one
richard	something really	come conclusion
encouraging	pretty sure	making friends
frequency	another one	people could

4.4.2 Dobór modeli

Pierwszym z zastosowanych algorytmów była wielomianowa regresja logistyczna zaimplementowana przy pomocy funkcji *LogisticRegression*[32] z biblioteki *scikit-learn.* W fazie eksperymentów zauważono wzrost wartości miary dokładności wraz ze zwiększaniem parametru *C* (Rys. 9), dlatego dobrano dla niego wartość 1. Regularyzacja jest wbudowana w metodę i wykonywana domyślnie, w przypadku tego problemu był to wariant L2. Podczas treningu jako funkcję kosztu przyjęto entropię krzyżową.

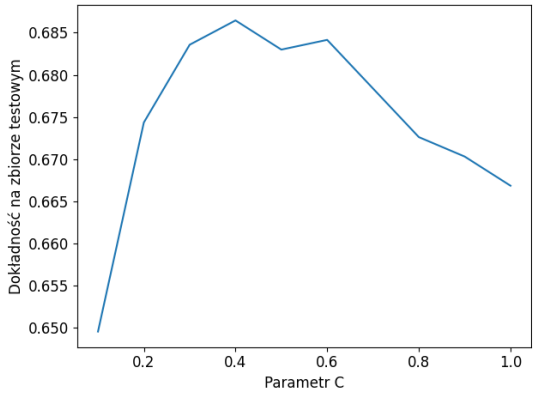
Rys. 9 – Wzrost dokładności regresji logistycznej na zbiorze testowym przy wzroście parametru C. Źródło: opracowanie własne



[32] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Kolejne podejście również opierało się na próbie znalezienia optymalnych separatorów klas. W jego przebiegu zastosowano liniowy klasyfikator SVM przystosowany dla potrzeb wieloklasowego zbioru, czyli z użyciem strategii jeden-przeciw-pozostałym. Posłużyła do tego klasa *LinearSVC* pochodząca z biblioteki *scikit-learn*[33]. Podobnie jak w przypadku regresji liniowej, przeprowadzono analizę parametru C, jednak na Rys. 10 można zauważyć, że tendencja w tym wypadku nie była monotoniczna. Jako optymalną wartość wybrano 0.4. Ponadto, jako funkcję kosztu wybrano kwadrat funkcji zawiasowej, który dobrze sprawdza się w treningu maszyn wektorów nośnych w zadaniu optymalizacyjnym znajdywania maksymalnego marginesu. Podczas treningu stosowano regularyzację L2.

Rys. 10 – Dokładność klasyfikatora SVM na zbiorze testowym przy wzroście parametru C. Źródło: opracowanie własne

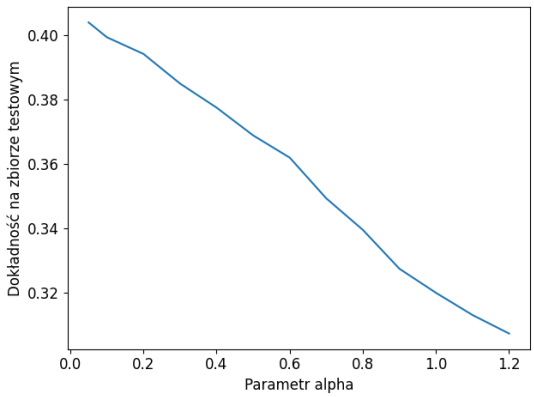


Następnym tradycyjnym modelem uczenia maszynowego wybranym do klasyfikacji szesnastu typów osobowości był wielomianowy naiwny klasyfikator bayesowski w implementacji *MultinomialNB*[34]. Wzrost parametru wygładzania addytywnego α był jednoznaczny ze zmniejszaniem się dokładności modelu, dlatego ustawiono jego wartość na niewielką, równą 0.1. Zaobserwowane zjawisko przedstawiono na Rys. 11.

Z komentarzem [MG14]: Kropki czy przecinki jako separatory dziesiętne?

[33] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[34] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

Rys. 11 – Spadek dokładności wielomianowego naiwnego klasyfikatora bayesowskiego na zbiorze testowym przy wzroście parametru alpha. Źródło: opracowanie własne



Ostatnim zaproponowanym algorytmem było ekstremalne wzmocnienie gradientowe. W tym celu wytrenowano klasyfikator *XGBClassifier* wywodzący się z modułu *XGBoost*[35]. Model automatycznie rozpoznał naturę klasyfikacyjną zadania i obliczał prawdopodobieństwo należenia próbek do więcej niż dwóch klas. Strojenie pary parametrów, czyli maksymalnej głębokości drzewa na zbiorze $\{3, 6, 9\}$ oraz lambdy związanej z regularyzacją L2 na zbiorze $\{0.1, 0.5, 1\}$ zakończyło się wyłonieniem najlepszej pary wartości – odpowiednio liczby 6 i 1.

4.5 Modelowanie z zastosowaniem sieci neuronowych

4.5.1 Kodowanie danych

Przygotowanie reprezentacji numerycznej tekstu dla modeli sieci neuronowych rozpoczęto od stworzenia słownika zawierającego wyrazy występujące we wstępnie przygotowanych i oczyszczonych postach zbioru treningowego. Spośród ponad 108 000 słów wykrytych przez funkcję *Tokenizer*[36], 100 000 z nich zostało uwzględnionych. Zastosowanie metody *fit_on_texts* zdeterminowało strategię budowy słownika opartej na częstości występowania wyrazów. To znaczy, że im dane słowo pojawiało się w próbkach częściej, tym znalazło się bliżej początku listy rekordów i tym niższy otrzymało indeks (Rys. 12). Po przygotowaniu

[35] https://xgboost.readthedocs.io/en/stable/get_started.html
[36] https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

opisanej struktury, przeprowadzono na jej podstawie proste mapowanie próbek do reprezentacji liczbowej (Rys. 13).

Rys. 12 – Pierwsze rekordy słownika z użyciem metody fit_on_texts klasy Tokenizer. Źródło: opracowanie własne

```
1 : [period]                    7 : think
2 : [ellipsis]                  8 : people
3 : [question_mark]             9 : one
4 : [number]                    10 : know
5 : [exclamation_mark]          11 : really
6 : like                        12 : would
                                ...
```

Rys. 13 – Przykładowe fragmenty próbek i ich reprezentacji numerycznej. Źródło: opracowanie własne

	post_cleaned	post_tokenized
2710	[care, others, tattoos, would, never, get, tat...	[164, 92, 3977, 12, 36, 14, 3561, 1, 164, 26, ...
5029	[momento, mori, [PERIOD], often, make, mistake...	[37877, 27522, 1, 131, 34, 1355, 412, 669, 21,...
3413	[thoughts, erik, better, known, phantom, opera...	[263, 15179, 82, 505, 6910, 4196, 43, 155, 841...

Najdłuższy ze scalonych postów zawierał 1081 tokenów numerycznych. Przy medianie długości wynoszącej 761, nie uznano potrzeby przycinania próbek. Uzupełniono zerami od lewej strony krótsze listy tokenów przy użyciu funkcji *pad_sequences*[37].

4.5.2 Sieć rekurencyjna z komórką GRU

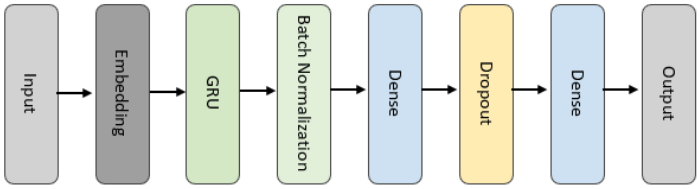
Neuronowe sieci rekurencyjne są szczególnie ukierunkowane na przetwarzanie sekwencji. Posiadają one unikalne struktury umożliwiające zapamiętywanie cech poprzednich elementów w szeregu, a przez to, w przypadku tekstu, wychwytujące kontekst i relacje słów w zdaniach. Ze względu na to należy pamiętać, że zachowanie kolejności poszczególnych wyrazów w próbce tekstowej jest kluczowe dla skutecznej klasyfikacji.

Architektura pierwszego zaproponowanego modelu sieci rekurencyjnej składała się z warstwy *Embedding*[38] osadzającej słowa w przestrzeni 64-wymiarowej, uwzględniając ich podobieństwa znaczeniowe. Ten segment można traktować jako dopełnienie dotychczasowych działań w kontekście tworzenia funkcjonalnej reprezentacji numerycznej

[37] https://www.tensorflow.org/api_docs/python/tf/keras/utils/pad_sequences
[38] https://keras.io/api/layers/core_layers/embedding/

próbek. Kolejnym komponentem modelu była pojedyncza komórka *GRU*[39] zawierająca 512 jednostek wraz z warstwą normalizującą *BatchNormalization*[40]. Po niej wystąpiła warstwa gęsta *Dense*[41] złożona z 128 neuronów z funkcją aktywacyjną ReLU, następnie mechanizm porzucania, czyli segment *Dropout*[42] odrzucający losowo 30% neuronów podczas każdego kroku treningu. Ostatnią warstwę stanowiła wyjściowa warstwa gęsta o 16 neuronach z funkcją aktywacyjną Softmax. Schemat opisanej architektury został zobrazowany na Rys. 14.

Rys. 14 – Schemat architektury rekurencyjnej sieci neuronowej z warstwą GRU. Źródło: opracowanie własne



Entropia krzyżowa w wariancie dla klasyfikacji wieloklasowej odgrywała rolę funkcji kosztu podczas treningu, a algorytm *Adam*[43] o domyślnych parametrach został wybrany do optymalizacji. Jako metrykę oceny skuteczności modelu obrano implementację dokładności *CategoricalAccuracy*[44], oraz *F1Score*[45] z uśrednianiem *macro*.

W ramach eksperymentów na opisanej architekturze sieci rekurencyjnej, podjęto próbę dostrajania jej wybranych hiperparametrów za pomocą metody *RandomSearch* pochodzącej z biblioteki *KerasTuner*[46]. Spośród wyszczególnionych w drugiej kolumnie Tabeli 6 propozycji parametrów i zasugerowanych zakresów wartości, mechanizm wybrał jako najlepszy zestaw cech przedstawiony w kolumnie trzeciej. Przedstawiona w dalszej części pracy analiza wyników wykazała, że zaproponowany zoptymalizowany model nie cechował się jednak lepszą skutecznością od swojego poprzednika.

[39] https://keras.io/api/layers/recurrent_layers/gru/
[40] https://keras.io/api/layers/normalization_layers/batch_normalization/
[41] https://keras.io/api/layers/core_layers/dense/
[42] https://keras.io/api/layers/regularization_layers/dropout/
[43] https://keras.io/api/optimizers/adam/
[44] https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalAccuracy
[45] https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/F1Score
[46] https://keras.io/keras_tuner/

Tabela 6 – Wartości proponowane oraz wybrane jako najlepsze podczas strojenia wybranych hiperparametrów sieci rekurencyjnej. Źródło: opracowanie własne

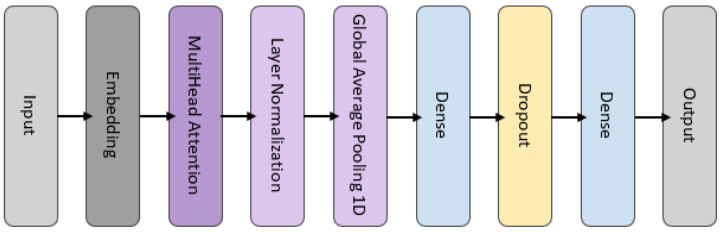
Opis parametru	Propozycje wartości	Najlepszy wybór
Liczba jednostek warstwy GRU	$\{128, 256, 512\}$	128
Liczba neuronów w pierwszej warstwie gęstej	$\{64, 128, 256\}$	128
Współczynnik regularyzacji L2 jądra pierwszej warstwy gęstej	$[0.0001, 0.01]$	≈ 0.00828
Wskaźnik uczenia optymalizatora Adam	$[0.0001, 0.01]$ Domyślnie 0.001	≈ 0.00898

4.5.3 Sieć neuronowa z mechanizmem Attention

Podobnie jak w przypadku sieci rekurencyjnej przeprowadzono mapowanie słów do przestrzeni 64-wymiarowej przy pomocy warstwy *Embedding*. Następnie zastosowano czteromodułowy mechanizm Attention jako obiekt klasy *MultiHeadAttention*[47]. Ponieważ warstwa ta przyjmowała jako komponenty *query*, *key* i *value* te same wartości, czyli tensor wyjściowy segmentu mapującego, można nazwać ją rozszerzonym na kilka równolegle przetwarzających procesów wariantem mechanizmu Self-Attention. Kolejnym elementem sieci była warstwa normalizująca *LayerNormalization*[48], a następnie procedura *GlobalAveragePooling1D*[49] w celu zmniejszenia wymiarowości wyjścia poprzez uśrednienie sąsiednich wartości. Po niej nastąpiła klasyczna warstwa gęsta składająca się z 32 neuronów, o funkcji aktywacyjnej zdefiniowanej jako ReLU. Nałożono na nią regularyzację L2[50] o współczynniku 0.01. Na zakończenie dobudowano mechanizm porzucania, pomijający losowo 30% neuronów i gęstą warstwę wyjściową z aktywacją Softmax. Rys. 15 ilustruje schemat architektury zaproponowanego modelu.

[47] https://keras.io/api/layers/attention_layers/multi_head_attention/
[48] https://keras.io/api/layers/normalization_layers/layer_normalization/
[49] https://keras.io/api/layers/pooling_layers/global_average_pooling1d/
[50] https://keras.io/api/layers/regularizers/ - l2-class

Rys. 15 – Schemat architektury sieci neuronowej z mechanizmem Attention. Źródło: opracowanie własne



W wypadku tego modelu, tak jak poprzedniego, jako funkcję kosztu wybrano kategorialną entropię krzyżową. Jednak tym razem wskaźnik uczenia został zmniejszony do 0.0001 w optymalizatorze Adam w celu skuteczniejszego śledzenia przebiegu treningu. Efektywność modelu mierzono za pomocą metryki dokładności.

4.5.4 Sieć neuronowa z użyciem modelu językowego BERT

Ostatnim uruchomionym na potrzeby tej pracy modelem była sieć neuronowa bazująca na modelu językowym BERT, zaimplementowana według oficjalnej instrukcji *Tensorflow*[51]. Stworzenie reprezentacji wektorowej próbek tekstu za pomocą pretrenowanej na większym zbiorze danych sieci polega na dopasowaniu jej do specyfiki analizowanego zbioru danych poprzez dostrojenie jej parametrów w procesie treningu.

Z komentarzem [MG15]: Dodano

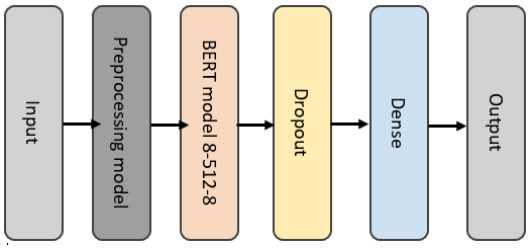
Warto dodać, że w przeciwieństwie do pozostałych zaimplementowanych architektur opartych na sieciach neuronowych, w tym przypadku dane tekstowe nie były przygotowane według schematu opisanego w Rozdziale 4.5.1. Zamiast tego, zastosowano gotowe rozwiązanie w postaci modelu przetwarzania wstępnego[52] modułu *tensorflow_hub* dedykowanego do zastosowania z wybranym modelem BERT.

Zaimplementowany model składał się z kilku segmentów. Po bloku przetwarzania wstępnego umieszczono zmniejszony wariant bazowej architektury BERT[53] zaczerpnięty z zasobów biblioteki *tensorflow_hub*. Struktura tego segmentu była analogiczna do oryginalnej

[51] https://www.tensorflow.org/text/tutorials/classify_text_with_bert
[52] Kod źródłowy zastosowanego modelu przetwarzania wstępnego: https://www.kaggle.com/models/tensorflow/bert/frameworks/tensorFlow2/variations/en-uncased-preprocess/versions/3?tfhub-redirect=true
[53] Kod źródłowy zastosowanego modelu językowego BERT: https://www.kaggle.com/models/tensorflow/bert/frameworks/tensorFlow2/variations/bert-en-uncased-l-8-h-512-a-8/versions/1?tfhub-redirect=true

z tą różnicą, że obsługiwała mniejszą liczbę parametrów, dzięki czemu trening przebiegał szybciej. Wybrana wersja była oparta na stosie jedynie ośmiu bloków koderów o wymiarowości 512 i ośmiomodułowym mechanizmie Attention. Zakodowany tekst przechodził następnie przez blok porzucający losowo 10% neuronów w kolejnych krokach treningu, a następnie był kierowany do gęstej warstwy wyjściowej zawierającej 16 neuronów i funkcję aktywacyjną Softmax. Schemat opisanej architektury przedstawiono na Rys. 16. W tej architekturze działa domyślnie optymalizator Adam.

Rys. 16 – Schemat architektury sieci neuronowej z użyciem modelu językowego BERT. Źródło: opracowanie własne



4.6 Środowisko eksperymentalne

Eksperymenty opisane w tej pracy zostały przeprowadzone za pomocą języka programowania Python w wersji 3.9. Jest to szybko rozwijające się, elastyczne narzędzie, szczególnie popularne w środowisku analitycznym, sprawdzające się w różnorodnych zadaniach, między innymi implementacji zagadnień uczenia maszynowego. Python jest obecnie jednym z najatrakcyjniejszych języków programowania wysokiego poziomu ze względu na czytelność jego składni oraz zapewnienie dostępu do wielu stabilnych i funkcjonalnych bibliotek. Jako środowisko pracy posłużyła interaktywna platforma Jupyter Notebook będąca elementem pakietu *Anaconda*[54].

W procesie przygotowania danych fundamentalną rolę odegrał moduł przeznaczony do analizy i manipulacji danymi *pandas*[55] 2.1 oraz biblioteka do operacji naukowych *NumPy*[56] 1.26. Budowanie wykresów opierało się natomiast na interfejsie *matplotlib.pyplot*[57]. Do

[54] https://www.anaconda.com/
[55] https://pandas.pydata.org/pandas-docs/version/2.1/index.html
[56] https://numpy.org/doc/stable/index.html
[57] matplotlib.pyplot — Matplotlib 3.5.3 documentation

implementacji klasycznych modeli uczenia maszynowego posłużyły głównie narzędzia analityczne modułu *scikit-learn*[58]. Sieci neuronowe były inicjowane i trenowane przy pomocy interfejsu *Keras*[59] biblioteki *Tensorflow* 2.10. Biblioteka ta operuje na tensorach, czyli obiektach matematycznych będących uogólnieniami pojęcia wektorów[60].

Z komentarzem [MG16]: W rozdziale 4.5.4 znajduje się jedyne użycie słowa „tensor", jednak tu wyjaśnienie lepiej się wpasowało w treść.

Użytkowany komputer charakteryzował się następującą specyfikacją:

- Procesor: 11th Gen Intel® Core™ i7 o taktowaniu 2.30 GHz i pamięcią podręczną Smart Cache o pojemności 24 MB,
- Karta graficzna: NVIDIA GeForce RTX 3050 z 2560 rdzeniami CUDA i 4 GiB pamięci RAM,
- Pamięć RAM: 32 GB o taktowaniu 3200 MHz,
- System operacyjny: Windows 11 Home 64-bitowy.

Niektóre procesy obliczeniowe zostały przeniesione na kartę graficzną dzięki dostępności architektury CUDA, co znacznie obniżyło łączny czas przeznaczony na trening modeli.

[58] https://scikit-learn.org/stable/
[59] https://keras.io/2.15/api/
[60] https://pl.wikipedia.org/wiki/Tensor, dostęp z dnia 05.04.2024 r.

5 Wyniki

5.1 Wieloklasowość

Wieloklasowość można określić jako występowanie wśród etykiet zbioru danych przeznaczonego do klasyfikacji więcej niż dwóch wartości.

Jak wynika z wcześniejszego przeglądu literatury, popularnym podejściem do klasyfikacji szesnastu typów osobowości modelu MBTI jest rozbicie problemu na cztery podzadania natury binarnej, co jest możliwe ze względu na specyfikę testu, którego celem jest przypisanie uczestnika do jednej wartości w każdej z czterech par istniejących kategorii. W tej pracy, nacisk był jednak kładziony na uzyskanie satysfakcjonujących wyników klasyfikacji dla zadania w jego pierwotnej formie, nie uwzględniając wspomnianych uproszczeń. Pojawiło się pytanie, czy jest to możliwe i jakie modele sprawdzą się do tego najlepiej.

Ze względu na wieloklasowy charakter problemu, większość testowanych modeli nie wystąpiło w swojej podstawowej postaci, a z pewnymi modyfikacjami. Przykładem mogą być ostatnie warstwy sieci neuronowych zawierające po 16 neuronów i funkcje aktywacji Softmax, zamiast jednej komórki wyjściowej aktywowanej funkcją Sigmoid – kombinacji typowej przy klasyfikacji binarnej. W tradycyjnych modelach uczenia maszynowego odpowiedzią na istnienie w zbiorze etykiet więcej niż dwóch klas może być użycie nie jednego, a grupy modeli konkretnego rodzaju w ramach podejścia jeden-przeciw-pozostałym, czy na przykład zastosowanie wielomianowego wariantu algorytmu.

Istotnym punktem odniesienia, aby określić czy model wychwytuje indywidualne wzorce na przestrzeni klas, jest wartość dokładności losowej klasyfikacji. Wyznaczenie wspomnianego współczynnika opiera się na wykonaniu prostego działania (49). Jeśli dla każdej próbki zbioru testowego etykieta zostałaby wylosowana, przy czym prawdopodobieństwo przypisania każdej z K klas byłoby takie samo, przy dodatkowym założeniu, że zbiór testowy był wystarczająco duży, procent odpowiednio zaklasyfikowanych próbek oscylowałby właśnie wokół poziomu losowego P_{losowy}.

$$P_{losowy} = \frac{100\%}{K} \tag{49}$$

Z komentarzem [MG17]: Poprawione

Dla omawianego w tej pracy problemu badawczego, przypisanie próbkom losowych wartości odpowiadającym szesnastu etykietom skutkowałoby uzyskaniem dokładności

klasyfikacji na poziomie około 6.25%. Jest to wartość, którą należy bezwzględnie przekroczyć, aby uznać, że model uczenia maszynowego lub sieć neuronowa są użyteczne.

5.2 Rezultaty eksperymentów

Poniżej przedstawiono wartości metryk ewaluacyjnych dla poszczególnych modeli. Ze względu na to, że miara dokładności jest na ogół niewystarczająca do otrzymania pełnego obrazu sprawności modeli, szczególnie w przypadku niezbalansowanego zbioru danych, posługiwano się również wskaźnikiem F_1 w wariancie *macro,* zwracającym średnią wartości F_1 obliczonych dla poszczególnych klas, dzięki czemu każda klasa jest traktowana równoważnie, niezależnie od liczności należącej do niej próbek.

Wszystkie modele były trenowane na 80% danych, a testowane na pozostałych 20%, przy czym rozkład próbek na przestrzeni klas był podobny w obu zbiorach. Wartości metryk były również śledzone na zbiorze treningowym między innymi w ramach detekcji przetrenowania.

W przypadku klasycznych modeli uczenia maszynowego rozpatrywane były unigramy, bigramy, a także kombinacja obu tych podejść do tworzenia worka słów. W prezentowanych poniżej wynikach zestawiono ze sobą rezultaty zastosowania każdej z tych trzech strategii.

Tabela 7 – Porównanie dokładności klasyfikacji klasycznych modeli uczenia maszynowego odpowiednio dla zbioru testowego i treningowego w zależności od strategii tworzenia worka słów. Źródło: opracowanie własne

	Unigramy		Unigramy i bigramy		Bigramy	
	testowy	treningowy	testowy	treningowy	testowy	treningowy
Regresja logistyczna	62.94%	72.61%	62.36%	72.75%	49.05%	71.79%
Liniowy SVM	**68.65%**	**89.16%**	**68.18%**	**89.55%**	**50.26%**	**94.45%**
Wielomianowy klasyfikator bayesowski	39.94%	53.62%	43.29%	54.55%	42.82%	65.52%
Ekstremalne wzmocnienie gradientowe	**69.16%**	**100%**	**68.41%**	**100%**	**52.80%**	**99.96%**

Z komentarzem [MG18]: Dodano, w Tabeli 8 też

W Tabeli 7 przedstawiono zaokrąglone wartości dokładności otrzymanych w fazie testowania poszczególnych modeli. Każda komórka zawiera procent poprawnych klasyfikacji uzyskany przez dany algorytm odpowiednio w zbiorze testowym i treningowym. Strategia tworzenia worka słów dająca najlepszy wynik na zbiorze testowym została w każdym wierszu zaznaczona zielonym kolorem. Tym samym, najbardziej tradycyjna metoda przypisywania tokenów jedynie pojedynczym słowom sprawdziła się najlepiej dla trzech z czterech modeli, choć wyniki po fuzji z bigramami były jej bardzo bliskie. Rozpatrzenie zarówno unigramów jak i bigramów przyniosło natomiast podwyższenie skuteczności probabilistycznego klasyfikatora bayesowskiego o 3.35 punkta procentowego na zbiorze testowym.

Pogrubioną czcionką zaznaczono w Tabeli 7 wyniki należące do dwóch najlepiej sprawdzających się modeli w całym zestawieniu, niezależnie od zawartości słownika. Liniowy klasyfikator SVM oraz ekstremalne wzmocnienie gradientowe uzyskały podobne wartości. Maksymalna skuteczność klasyfikacji pierwszego z nich to 68.65% na zbiorze testowym i 89.16% na zbiorze treningowym. Drugi z nich osiągnął niewiele lepszy wynik na zbiorze testowym równy 69.16%, natomiast zbiór treningowy zaklasyfikował bezbłędnie.

Tabela 8 – Porównanie wartości wskaźnika F_1 klasycznych modeli uczenia maszynowego dla klasyfikacji odpowiednio zbioru testowego i treningowego w zależności od strategii tworzenia worka słów. Źródło: opracowanie własne

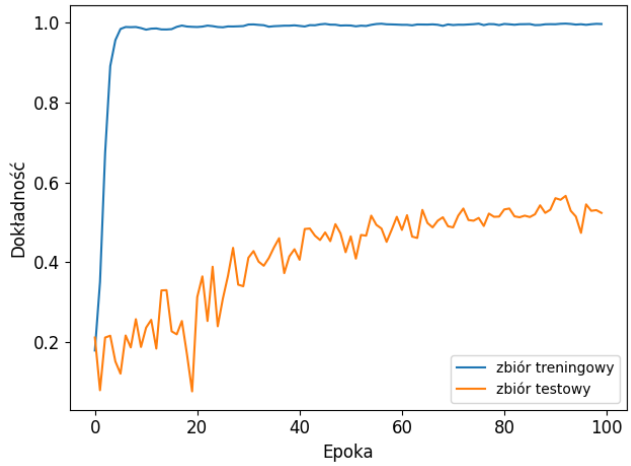
	Unigramy		Unigramy i bigramy		Bigramy	
	testowy	treningowy	testowy	treningowy	testowy	treningowy
Regresja logistyczna	35.38%	45.87%	34.97%	45.59%	21.77%	38.48%
Liniowy SVM	52.21%	86.46%	52.66%	87.01%	29.44%	94.71%
Wielomianowy klasyfikator bayesowski	13.41%	24.70%	15.53%	25.65%	15.58%	50.25%
Ekstremalne wzmocnienie gradientowe	**56.49%**	**100%**	**55.53%**	**100%**	**33.30%**	**99.98%**

W celu bardziej kompleksowej analizy efektywności zastosowanych klasycznych modeli uczenia maszynowego, w Tabeli 8 zaprezentowano uzyskane podczas testów uśrednione wartości wskaźnika F_1 w zaokrągleniu. Można zauważyć, że wnioski po analizie metryki dokładności częściowo pokrywają się z tendencjami widocznymi w tym zestawieniu. Wielomianowy naiwny klasyfikator bayesowski ponownie okazał się być najsłabszym rozwiązaniem, uzyskując maksymalny wskaźnik F_1 równy 15.58%. Przyjmując strategię unigramów, model ekstremalnego wzmocnienia gradientowego okazał się być znów najbardziej skuteczny pośród całego zestawienia z wynikiem 56.49% na zbiorze testowym. Można zauważyć, że zachodziła też bardziej wyraźna różnica w porównaniu jego rezultatów z wartościami osiągniętymi przez liniowy klasyfikator SVM. Analiza zielonych pól nasuwa wniosek, że w zależności od priorytetów zadania, zastosowanie bigramów może być również dobrym wyborem.

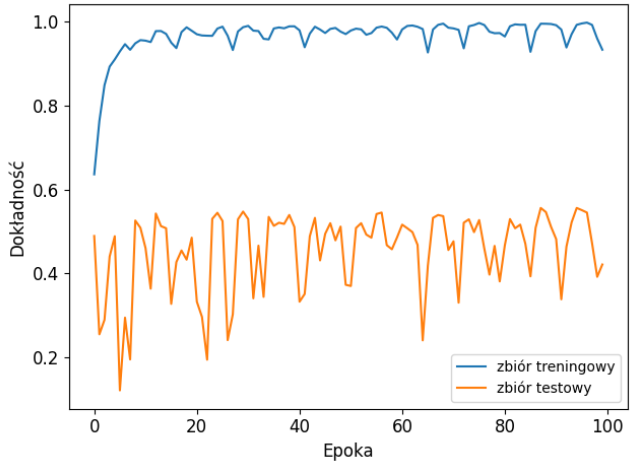
Przechodząc do wyników modelowania sieci neuronowych, jak zostało opisane w Rozdziale 4.5.1, były one trenowane na danych zakodowanych z uwzględnieniem kolejności występowania wyrazów, a nie za pomocą metod statystycznych.

Sieć rekurencyjna z warstwą GRU była trenowana przez 100 epok, bez specyfikacji wielkości paczek danych, podobnie jak jej zoptymalizowany przez *KerasTuner* wariant. Na Rys. 17 i Rys. 18 można porównać przebieg wartości miary dokładności podczas trwania treningów każdego z tych modeli. Metryki sieci niezoptymalizowanej charakteryzują się stabilniejszym trendem rosnącym niż sieci zasugerowanej przez mechanizm dostrajający.

Rys. 17 – Wykres dokładności sieci rekurencyjnej z komórką GRU. Źródło: opracowanie własne

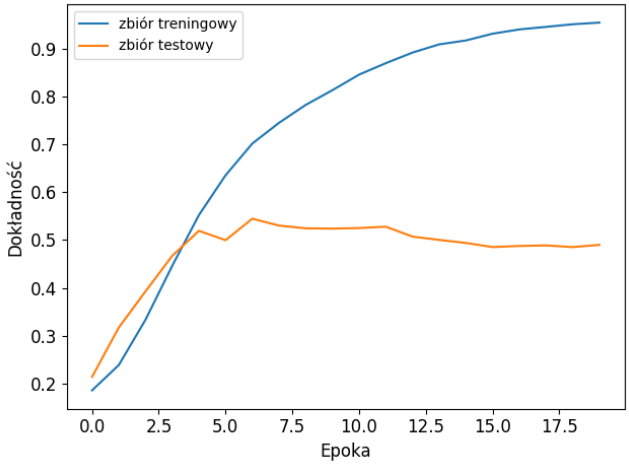


Rys. 18 – Wykres dokładności sieci rekurencyjnej z komórką GRU po strojeniu hiperparametrów. Źródło: opracowanie własne



Sieć neuronowa z modułem Attention uczyła się przez 20 epok, jednorazowo przetwarzając ośmioelementową podgrupę próbek. Na Rys. 19 można zaobserwować zachowanie miary dokładności na przestrzeni tych epok dla zbioru treningowego oraz testowego. Wartości opisujące ten drugi zestaw danych osiągają swoje maksimum po szóstej epoce, a następnie przechodzą w trend spadkowy.

Rys. 19 – Wykres dokładności sieci neuronowej z modułem Attention. Źródło: opracowanie własne



Głęboka sieć oparta na zastosowaniu modelu językowego BERT trenowana była z użyciem ośmioelementowych paczek danych przez 200 epok. Mimo długiego czasu strojenia nie osiągnęła wyników porównywalnych do innych przedstawionych w pracy modeli, a postępy uczenia nie były wyraziste w kontekście wartości miary dokładności. Na Rys. 20 przedstawiono wyniki tej metryki w procesie treningu, operowały one w okolicach 17%. Można zauważyć zmniejszającą się z czasem amplitudę wyników dla zbioru treningowego przy równoczesnym trendzie wzrostowym, zmiany są jednak zbyt mało zauważalne, aby model uznać za coraz skuteczniejszy w klasyfikacji przypadkowych danych. Z tego powodu, opisana sieć neuronowa została wykluczona z poniższego porównania wyników.

Rys. 20 – Wykres dokładności sieci neuronowej z użyciem modelu językowego BERT.
Źródło: opracowanie własne

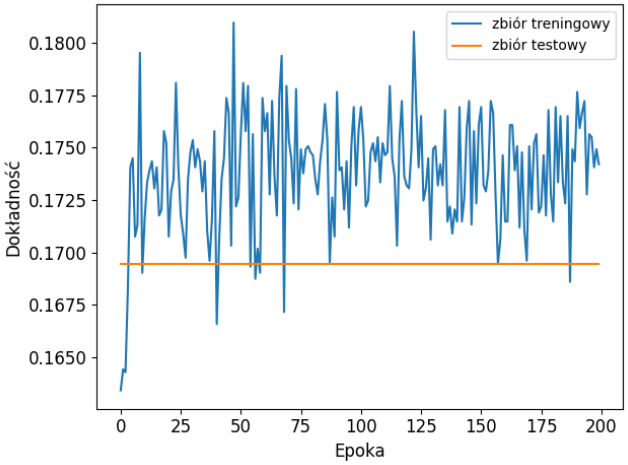


Tabela 9 przedstawia porównanie wyników inferencji zbioru testowego i treningowego z użyciem opisanych modeli sieci neuronowych. Spośród dwóch sieci rekurencyjnych z warstwą GRU lepsze wyniki uzyskała sieć niezoptymalizowana. Poradziła sobie ona również lepiej ze swoim zadaniem niż sieć z mechanizmem Attention, jednak wciąż nie tak dobrze jak niektóre tradycyjne modele uczenia maszynowego opisane wcześniej.

Tabela 9 – Porównanie wskaźników oceny zastosowanych modeli opartych na sieciach neuronowych.
Źródło: opracowanie własne

		Zbiór testowy	Zbiór treningowy
RNN z warstwą GRU	Dokładność	**52.33%**	**99.47%**
	Wskaźnik F_1	**32.67%**	**99.48%**
	Wskaźnik F_1 na przestrzeni klas	63.72% 41.57% 54.92% 58.47% 25.35% 25.00% 59.97% 57.39% 20.00% 33.04% 53.85%	99.91% 99.55% 98.63% 99.89% 100% 100% 99.45% 99.72% 99.54% 99.81% 99.62%

		0% 29.41% 0% 0% 0%	96.20% 99.30% 100% 100% 100%
Zoptymalizowana RNN z warstwą GRU	Dokładność	48.65%	88.78%
	Wskaźnik F_1	29.87%	79.58%
	Wskaźnik F_1 na przestrzeni klas	40.51% 14.77% 60.09% 52.50% 22.64% 25.64% 62.90% 49.61% 28.07% 45.16% 34.92% 31.03% 0% 0% 10% 0%	88.54% 71.66% 91.91% 92.81% 85.80% 69.46% 97.18% 89.40% 75.39% 85.94% 91.73% 90.00% 62.07% 28.57% 71.43% 81.36%
Sieć z modułem Attention	Dokładność	48.99%	97.58%
	Wskaźnik F_1	26.71%	76.32%
	Wskaźnik F_1 na przestrzeni klas	55.71% 43.84% 54.79% 53.07% 17.50% 3.33% 68.24% 43.14% 18.35% 36.21% 7.75% 17.48% 8.00% 0% 0% 0%	99.58% 100% 99.90% 99.83% 99.46% 95.87% 99.76% 98.81% 87.50% 98.72% 87.77% 89.94% 52.53% 0% 0% 11.43%

Warto zwrócić uwagę na to, że przedstawione w poszczególnych zestawieniach wysokości wskaźnika F_1 zostały uzyskane poprzez uśrednienie jego wartości obliczonych osobno dla każdej z szesnastu klas – przykładowy ich rozkład można zaobserwować w Tabeli 9. W zależności od modelu dystrybucja ta się różniła, jednak podstawową tendencją, jaką można

było zauważyć podczas samej pracy badawczej, było to, że dla zbioru testowego miara F_1 przyjmowała wartości 0% lub jej bliskie najczęściej dla klas o najmniejszej liczności.

Warto zastanowić się, jak należy interpretować takie zjawisko. Z jednej strony, należy pamiętać, że poziom zaprezentowanego uśrednionego wskaźnika bez wątpienia nie wyklucza istnienia dysonansu między poszczególnymi klasami w aspekcie wysokości precyzji i czułości. To powszechny przypadek, że model radzi sobie lepiej z pewnymi klasami, a z pewnymi gorzej, i na tym budowana jest ocena jego generalnej skuteczności. Warto wspomnieć, że w procesie uśredniania nie zastosowano wag, właśnie w celu podkreślenia, że umiejętność wykrywania każdego typu osobowości jest równie ważna dla modelu.

Podchodząc do wspomnianej obserwacji z innej strony, mała ilość próbek należących do danej klasy w zbiorze testowym, co bezpośrednio wywodzi się z problemu niezbalansowanego zbioru danych, może prowadzić do zniekształcenia wyników dotyczących ogólnej skuteczności modelu, ponieważ wylosowane do tego podzbioru próbki nie muszą odzwierciedlać w pełni natury klasy, albo mogą być swego rodzaju punktami odstającymi.

5.3 Czasy treningów i inferencji

Porównanie czasów treningów oraz inferencji pojedynczych próbek dla poszczególnych modeli może być znaczące, aby uzyskać ich pełen obraz. W Tabeli 10 zebrano informacje na temat czasu uczenia poszczególnych algorytmów wyrażonego w sekundach. Warto podkreślić, że w przypadku tradycyjnych modeli uczenia maszynowego, podano czasy całego treningu, natomiast dla sieci neuronowych wyszczególniono liczbę sekund przypadającą na jedną epokę. Czas inferencji dla jednej próbki podano w milisekundach, jest to mediana czasów dla 100 predykcji, ponieważ często pierwsze przebiegi sieci po wytrenowaniu trwają o wiele dłużej niż późniejsze.

Z zestawienia jasno wynika, że ze względu na wiele epok treningi sieci neuronowych były najbardziej czasochłonne. Można również zauważyć, że generowanie predykcji trwało krócej w przypadku tradycyjnych modeli uczenia maszynowego. Co więcej, wśród tradycyjnych modeli czas potrzebny na nauczenie i przeprowadzenie inferencji ekstremalnego wzmocnienia gradientowego był znacznie wyższy od wymagań pozostałych trzech algorytmów. W przypadku sieci neuronowych, najdłuższym czasem treningu odznaczała się architektura z użyciem modelu językowego BERT, natomiast wygenerowanie predykcji najdłużej zajęło niezoptymalizowanej sieci rekurencyjnej z komórką GRU.

Tabela 10 – Czasy treningów i inferencji zaproponowanych modeli. Źródło: opracowanie własne

	Czas treningu	Czas inferencji dla jednej próbki
Regresja logistyczna (unigramy)	4.6243 s	0.1708 ms
Liniowy SVM (unigramy)	3.7798 s	0.1067 ms
Wielomianowy klasyfikator bayesowski (unigramy)	0.0842 s	0.1101 ms
Ekstremalne wzmocnienie gradientowe (unigramy)	228.3398 s	1.9728 ms
RNN z warstwą GRU	25 s/epoka	67.9033 ms
Zoptymalizowany RNN z warstwą GRU	15 s/epoka	66.9966 ms
Sieć z modułem Attention	34 s/epoka	60.1139 ms
Sieć z użyciem modelu językowego BERT	155 s/epoka	43.7212 ms

5.4 Uzyskane wyniki na tle innych badań

Zdecydowana większość przedstawionych wcześniej w przeglądzie literatury badań jest oparta na koncepcji rozbicia problemu badawczego na cztery pomniejsze zadania klasyfikacji binarnych. Jednym z pytań postawionych w tej pracy było, czy jest możliwość stworzenia klasyfikatora szesnastoklasowego, którego skuteczność będzie wysoka na tle wspomnianych prób mimo innego, wydawałoby się mniej obiecującego, podejścia. Warto wspomnieć, że artykuły poszczególnych autorów różniły się nie tylko pod względem modelowania, ale także przygotowywania danych, definicji pojedynczej próbki, a także ich podziału. Przedstawione wyniki są więc pewnym przybliżeniem efektów klasyfikacji uzyskanych przez dane modele i są jedynie orientacyjnym punktem porównawczym dla eksperymentów zaproponowanych w tej pracy. W Tabeli 11 przedstawiono źródła, których autorzy bazowali na tym samym zbiorze danych, opisanym w Rozdziale 1.4. W poszczególnych kolumnach ukazano podstawowe charakterystyki każdej strategii oraz uzyskane wyniki dokładności. W przypadku, kiedy autorzy podali jedynie cztery wyniki cząstkowe dla każdej z klasyfikacji binarnych na zbiorze testowym, zastosowano agregację poprzez wyliczenie ich iloczynu.

Można zauważyć, że jedynie jeden model spośród wszystkich przedstawionych w Tabeli 11 uzyskał skuteczność wykraczającą ponad 50%. Cztery inne podejścia przyniosły natomiast dokładność ponad 40%. Kilka modeli zaimplementowanych w ramach tej pracy zagwarantowało porównywalną lub większą skuteczność. Przykładem może być sieć rekurencyjna z warstwą GRU, która zaklasyfikowała poprawnie zbiór testowy w 52.33%. Spośród tradycyjnych modeli uczenia maszynowego, najlepszy z nich osiągnął dokładność na poziomie 69.16%, co okazuje się być bardzo dobrym, znaczącym wynikiem na tle eksperymentów innych autorów.

Tabela 11 – Wyniki modelowania zaproponowanego przez innych autorów. Źródło: opracowanie własne

Artykuł	Uwagi dotyczące strategii	Model	Dokładność w zbiorze testowym
Predicting Myers-Briggs Type Indicator with Text Classification (Hernandez & Knight, 2017)	Predykcje binarne. Wyniki zagregowane poprzez zastosowanie iloczynu.	Sieć rekurencyjna LSTM	8.73%
Survey Analysis of Machine Learning Methods for Natural Language Processing for MBTI Personality Type Prediction (Cui & Qi, 2017)	Predykcje dla 16 klas.	Prosty klasyfikator Softmax	17.00%
		Głęboka sieć neuronowa	23.00%
	Predykcje binarne. Wyniki zagregowane przez autorów.	Naiwny klasyfikator bayesowski	25.86%
		Klasyfikator SVM	32.60%
		Głęboka sieć neuronowa	38.00%
Data Science Final Project: Myers-Briggs Prediction (Antonio, et al., 2018)	Zbiór danych zbalansowany metodą SMOTE. Predykcje binarne. Wyniki uzyskane za pomocą 10-krotnej walidacji	Regresja logistyczna	22.61%
		Klasyfikator SVM	24.08%
		Las losowy	25.52%

	krzyżowej i zagregowane przez autorów.	Ekstremalne wzmocnienie gradientowe	25.32%
Improving Intelligent Personality Prediction using Myers-Briggs Type Indicator and Random Forest Classifier (Abidin, et al., 2020)	Predykcje binarne. Wyniki zagregowane przez autorów.	Regresja logistyczna	23.35%
		Klasyfikator SVM	16.94%
		kNN	**40.62%**
		Las losowy	**100.00%**
Machine Learning Approach to Personality Type Prediction Based on the Myers–Briggs Type Indicator (Amirhosseini & Kazemian, 2020)	Predykcje binarne. Wyniki zagregowane poprzez zastosowanie iloczynu. Dokładność określona przez autorów określa skuteczność klasyfikacji typu osobowości użytkownika, a nie klasy posta – metoda ta została opisana w jednym ze starszych artykułów (Hernandez & Knight, 2017).	Ekstremalne wzmocnienie gradientowe	31.73%
		Ekstremalne wzmocnienie gradientowe z konfiguracją głębokości drzewa	32.96%
Personality Type Based on Myers-Briggs Type Indicator with Text Posting Style by using Traditional and Deep Learning (Ontoum & Chan, 2022)	Predykcje binarne. Wyniki zagregowane przez autorów.	Naiwny klasyfikator bayesowski	**41.03%**
		Klasyfikator SVM	**41.97%**
		Rekurencyjna sieć neuronowa	**49.75%**

6 Podsumowanie

W ramach tej pracy przeprowadzono badania na temat klasyfikacji typów osobowości modelu MBTI na podstawie wpisów użytkowników z portalu społecznościowego. Podstawą eksperymentów był popularny zbiór danych *(MBTI) Myers-Briggs Personality Type Dataset*.

Dane tekstowe zostały wstępnie przetworzone poprzez złączenie w jedno postów każdej z osób, zmniejszenie liter, tokenizację, usunięcie słów bez znaczenia i flagowanie charakterystycznych elementów wpisów. Następnie podzielono próbki na zbiór treningowy oraz testowy w sposób stratyfikowany. Z użyciem algorytmu TF-IDF stworzono reprezentację numeryczną próbek, która posłużyła do treningu czterech tradycyjnych modeli uczenia maszynowego. Dla każdego z nich przeprowadzono trzy próby posługując się unigramami, bigramami, a także rozważając mieszane człony słownika. Dla porównania, dane osadzone w przestrzeni 64-wymiarowej były bazą dla treningu architektur sieci neuronowych. W ramach tej części, przetestowano skuteczność sieci rekurencyjnej z komórką GRU, jej odpowiednika po strojeniu hiperparametrów, a także modelu zawierającego czteromodułowy mechanizm Attention. Dodatkowo sprawdzono, jakie efekty przyniesie uczenie sieci neuronowej dla danych zakodowanych za pomocą pretrenowanego modelu językowego BERT. Następnie przeprowadzono ewaluację uzyskanych wyników oraz skonfrontowano je z wartościami metryk opublikowanymi w artykułach przez innych badaczy zajmujących się podobnym problemem badawczym. Ponadto, porównano czasy treningu i inferencji dla każdego z zaimplementowanych modeli.

Wśród przeprowadzonych eksperymentów wyłoniono algorytm ekstremalnego wzmocnienia gradientowego uruchomiony na danych przygotowanych techniką TF-IDF na bazie unigramów jako najskuteczniejsze z wszystkich przetestowanych metod rozwiązanie postawionego problemu badawczego. Wspomniany model osiągnął wartość dokładności na poziomie 69.16% na zbiorze testowym przy wskaźniku F_1 przyjmującym wartość 56.49%.

Cel pracy został osiągnięty. Eksperymenty dowiodły, że zadany problem badawczy może być skutecznie rozwiązany przy pomocy metod uczenia maszynowego i sztucznej inteligencji. Udowodniono, że dane tekstowe mogą być rzetelnym źródłem informacji, jeśli zostaną odpowiednio przygotowane i zakodowane. Przedstawiono szereg metod przetwarzania próbek oraz strategii ich modelowania. Uzyskany wynik jest obiecujący dla zadania klasyfikacji

typów osobowości na podstawie analizy postów internetowych, także na tle innych istniejących rozwiązań. Warto dodać, że problemy natury psychologicznej poddane analizie danych mogą charakteryzować się większymi nieścisłościami predykcji ze względu na różnorodność grupy badawczej i ludzką złożoność.

Stworzony model ma szansę posłużyć jako wsparcie pracy rekruterów biznesowych czy silnik funkcjonalnego dodatku podpiętego do portali społecznościowych i wspomagającego rozwój osobisty jego użytkowników.

Spis tabel

Spis rysunków

Bibliografia

Abidin, N. H. Z. i inni, 2020. Improving Intelligent Personality Prediction using Myers-Briggs Type Indicator and Random Forest Classifier. *International Journal of Advanced Computer Science and Applications*, Tom XI.

Amirhosseini, M. H. i Kazemian, H., 2020. Machine Learning Approach to Personality Type Prediction Based on the Myers-Briggs Type Indicator ®. *Multimodal Technologies and Interaction*, Tom IV.

Antonio, B. i inni, 2018. *Medium.* [Online]
Available at: https://medium.com/@bian0628/data-science-final-project-myers-briggs-prediction-ecfa203cef8

Ba, J. L., Kiros, J. R. i Hinton, G. E., 2016. Layer Normalization.

Bengio, Y., Simard, P. i Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, Tom V, pp. 157-166.

Chen, T. i Guestrin, C., 2016. XGBoost: A Scalable Tree Boosting System.

Cho, K. i inni, 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.

Cui, B. i Qi, C., 2017. Survey Analysis of Machine Learning Methods for Natural Language Processing for MBTI Personality Type Prediction.

Devlin, J., Chang, M.-W., Lee, K. i Toutanova, K., 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Fenner, M. E., 2020. *Uczenie maszynowe w Pythonie dla każdego.* Gliwice: Helion.

Géron, A., 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* Sebastopol: O'Reilly Media.

Géron, A., 2018. *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow : pojęcia, techniki i narzędzia służące do tworzenia inteligentnych systemów.* Gliwice: Helion.

Hernandez, R. i Knight, I. S., 2017. Predicting Myers-Briggs Type Indicator with Text Classification.

Hochreiter, S. i Schmidhuber, J., 1997. Long Short-term Memory. *Neural computation*, Tom IX, pp. 1735-1780.

Ioffe, S. i Szegedy, C., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

Jung, C. G., 2013. *Typy psychologiczne.* I red. Warszawa: KR.

Kingma, D. i Ba, J., 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*.

McCulloch, W. S. i Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, Tom V, pp. 115-133.

Ontoum, S. i Chan, J., 2022. Personality Type Based on Myers-Briggs Type Indicator with Text Posting Style by using Traditional and Deep Learning.

Raschka, S. i Mirjalili, V., 2019. *Python. Uczenie maszynowe.* II red. Gliwice: Helion.

Rosset, S., Zhu, J. i Hastie, T., 2004. Margin Maximizing Loss Functions. *Advances in Neural Information Processing Systems.*

Rumelhart, D. E., Hinton, G. E. i Williams, R. J., 1986. Learning Internal Representations by Error Propagation.

Skiena, S. S., 2017. *The Data Science Design Manual.* I red. Cham: Springer Cham.

Vaswani, A. i inni, 2023. Attention Is All You Need. *Neural Information Processing Systems*.