Obraz zawierający tekst, Czcionka, logo, Grafika

Opis wygenerowany automatycznie

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**

dodać

Słowa kluczowe:

**Abstract**

dodać

Keywords:

# Spis treści

[Spis treści 5](#_Toc161494749)

[1 Wprowadzenie 9](#_Toc161494750)

[1.1 Motywacja 9](#_Toc161494751)

[1.2 Cel i zakres pracy 10](#_Toc161494752)

[1.3 Model osobowości MBTI 11](#_Toc161494753)

[1.4 Zbiór danych 12](#_Toc161494754)

[2 Przegląd literatury 14](#_Toc161494755)

[3 Wstęp teoretyczny 16](#_Toc161494756)

[3.1 Przetwarzanie języka naturalnego 16](#_Toc161494759)

[3.2 Przygotowanie danych tekstowych 16](#_Toc161494760)

[3.3 Ekstrakcja dodatkowych cech 17](#_Toc161494761)

[3.4 Reprezentacja numeryczna tekstu 17](#_Toc161494762)

[3.4.1 Metody uwzględniające kolejność wyrazów 18](#_Toc161494763)

[3.4.2 Metody statystyczne 19](#_Toc161494764)

[3.5 Klasyczne modele uczenia maszynowego 19](#_Toc161494765)

[3.5.1 Wielomianowa regresja logistyczna 19](#_Toc161494766)

[3.5.2 Liniowy klasyfikator SVM 22](#_Toc161494767)

[3.5.3 Wielomianowy naiwny klasyfikator bayesowski 25](#_Toc161494768)

[3.5.4 Drzewo decyzyjne 26](#_Toc161494769)

[3.5.5 Ekstremalne wzmocnienie gradientowe 27](#_Toc161494770)

[3.6 Sztuczne sieci neuronowe 32](#_Toc161494771)

[3.6.1 Rekurencyjne sieci neuronowe 34](#_Toc161494772)

[3.7 Mechanizm Attention 39](#_Toc161494773)

[3.8 Trening i ewaluacja modeli 40](#_Toc161494774)

[3.8.1 Walidacja krzyżowa 40](#_Toc161494775)

[3.8.2 Sprawdzian prosty 40](#_Toc161494776)

[3.8.3 Funkcja kosztu – entropia krzyżowa 40](#_Toc161494777)

[3.8.4 Ewaluacja modelu 40](#_Toc161494778)

[4 Materiały i metody 42](#_Toc161494779)

[4.1 Wstępne przetwarzanie tekstu 42](#_Toc161494781)

[4.2 Dodatkowe charakterystyki numeryczne 44](#_Toc161494782)

[4.3 Podział danych 45](#_Toc161494783)

[4.4 Klasyfikacja za pomocą tradycyjnych modeli uczenia maszynowego 46](#_Toc161494784)

[4.4.1 Przygotowanie reprezentacji wektorowej 46](#_Toc161494785)

[4.4.2 Dobór modeli 47](#_Toc161494786)

[4.5 Modelowanie z zastosowaniem głębokich sieci neuronowych 50](#_Toc161494787)

[4.5.1 Kodowanie danych 50](#_Toc161494788)

[4.5.2 Sieć rekurencyjna z komórką GRU 51](#_Toc161494789)

[4.5.3 Sieć neuronowa z mechanizmem Attention 52](#_Toc161494790)

[4.6 Środowisko eksperymentalne 53](#_Toc161494791)

[5 Wyniki 55](#_Toc161494792)

[5.1 Wieloklasowość 55](#_Toc161494797)

[5.2 Rezultaty eksperymentów 56](#_Toc161494798)

[6 Podsumowanie 62](#_Toc161494799)

[7 Roboczy generator schematów 63](#_Toc161494804)

[Spis tabel 64](#_Toc161494805)

[Spis rysunków 65](#_Toc161494806)

[Wykaz symboli i oznaczeń 67](#_Toc161494807)

[Wykaz używanych skrótów 68](#_Toc161494808)

[Bibliografia 69](#_Toc161494813)

# Wprowadzenie

## Motywacja

Kwestionariusz Myers-Briggs powstał w połowie ubiegłego wieku i jest jednym z najbardziej rozpowszechnionych modeli osobowości w biznesie[[1]](#footnote-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]](#footnote-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]](#footnote-3). Tagi definiujące użytkownika z użyciem typów MBTI pojawiły się nawet w niektórych społecznościach platformy Reddit[[4]](#footnote-4), a także zostały wprowadzone do popularnych aplikacji randkowych, w celu poszukiwania jeszcze większej kompatybilności między potencjalnymi partnerami. Jedna z najbardziej popularnych stron internetowych[[5]](#footnote-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 dodatkową 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, tym bardziej, że wyniki testu potrafią wahać się w zależności od okoliczności i samopoczucia danej osoby w chwili wykonywania kwestionariusza[[6]](#footnote-6).

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

## 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.

## 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[[7]](#footnote-7).

Klasyfikacja MBTI opiera się na określeniu wartości dla czterech par kategorii[[8]](#footnote-8), opisanych w Tabeli 1.

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

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

## Zbiór danych

*(MBTI) Myers-Briggs Personality Type Dataset* to zbiór danych dostępny publicznie na platformie Kaggle[[9]](#footnote-9). Zawiera on 8675 wierszy, przy czym każdy z nich odpowiada ostatnim pięćdziesięciu postom udostępnionym przez danego użytkownika na forum PersonalityCafe[[10]](#footnote-10) 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).

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



# 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*.

[Więcej przykładów? 16 klasowość albo wyższe wyniki?]

# Wstęp teoretyczny



## 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.

## 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.

## 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.

## 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.

### 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 ich 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[[11]](#footnote-11).

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*).

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.

### 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. Algorytm nadaje wagi słowom – największą przypisuje rzadkim wyrazom, które w danej próbce wystąpiły często. Dopuszcza również traktowanie par, trójek lub większych zestawień słów, tak zwanych n-gramów, jako całości, co oznacza, że dla każdego n-gramu liczona jest wówczas pojedyncza reprezentacja.

[wstawić wzór TF-IDF]

## Klasyczne modele uczenia maszynowego

### Wielomianowa regresja logistyczna

Klasyczna regresja logistyczna 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 *(1).*

(1)

Funkcja logistyczna jest szeroko stosowaną w statystyce funkcją sigmoidalną, która przekształca dane wejściowe do wartości z przedziału . Można ją wyrazić prostym równaniem *(2).*

(2)

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

Uczenie modelu polega na znalezieniu takiego wektora parametrów , aby zwracane wartości były wysokie dla przykładów klasy pozytywnej, a niskie dla próbek klasy negatywnej. Odwrotne przypisania należy karać podczas treningu. Umożliwia to logarytmiczna funkcja straty (ang. *log loss*) *(3)*. Przyjęto oznaczenie jako liczbę próbek w zbiorze treningowym.

(3)

Minimalizację tej funkcji kosztu można przeprowadzić między innymi za pomocą metody wsadowego gradientu prostego operując na pochodnych cząstkowych obliczanych dla każdego parametru

W problemach wieloklasowych stosuje się wielomianowy wariant regresji logistycznej. W tym celu dla danej próbki należy najpierw obliczyć osobny wynik dla każdej klasy uwzględniając jej indywidualny wektor parametrów *(4).*

(4)

Kolejnym krokiem jest przekazanie wyników wszystkich klas dla danego przykładu do funkcji Softmax *(5)*, która na tej podstawie oblicza prawdopodobieństwa przynależności do każdej z klas.

(5)

Predykcja 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 *(6)* i stosuje się ją często w uczeniu modeli przeznaczonych do klasyfikacji wieloklasowej. Jako argument przyjmuje macierz parametrów modelu . Jeśli jednak dostępne są dwie klasy, warto zwrócić uwagę, że funkcja ta przekształca się w logarytmiczną funkcję straty *(3).* 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 .

(6)

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 *(7)* karci duże wartości wag poprzez dodanie dodatkowego obciążenia do funkcji kosztu. Warto podkreślić, że kary nie są nakładane na wyrazy wolne , ponieważ suma liczona jest od .

.

(7)

Implementacja regularyzacji polega na dodaniu członu regularyzacji do funkcji kosztu *(8)*. W zaprezentowanych równaniach oznacza parametr regularyzacji. Siła regularyzacji wzrasta wprost proporcjonalnie do jego wartości. W implementacjach programistycznych współczynnik ten może być zastąpiony parametrem , który wówczas należy interpretować jako odwrotność .

(8)

### 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 *(9)*, gdzie symbolizuje wektor współczynników, a oznacza wektor zmiennych niezależnych.

(9)

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 , gdzie ujemna wartość etykiety oznacza klasę negatywną, margines zostaje opisany wzorami *(10)*.

(10)

Długość wektora można z łatwością wyznaczyć *(11).*

(11)

Odejmując od siebie równania układu *(10),* a następnie normalizując wynik z użyciem długości wektora *(11)* powstaje równanie *(12)*, którego lewa strona można zinterpretować jako odległość między negatywną i pozytywną hiperpłaszczyzną, czyli poszukiwany margines*.* Algorytm SVM stara się znaleźć maksymalną wartość wyrażenia , co jest równoznaczne z wyznaczeniem największego możliwego marginesu.

(12)

Podczas optymalizacji, musi on jednak uwzględnić dodatkowy warunek, że każda próbka powinna być poprawnie klasyfikowana, to znaczy zależnie od opisującej ją etykiety , powinna leżeć na lub za wyznaczoną hiperpłaszczyzną marginesu *(13).*

(13)

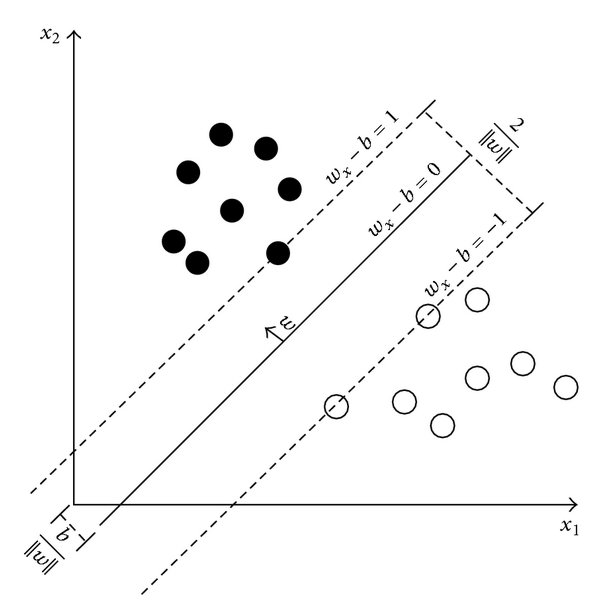
Powyższy warunek może być zapisany w krótszej postaci *(14).*

(14)

W rzeczywistości zamiast minimalizacji stosuje się minimalizację wyrażenia ze względu na jego różniczkowalność w punkcie i wygodną w przetwarzaniu pochodną.

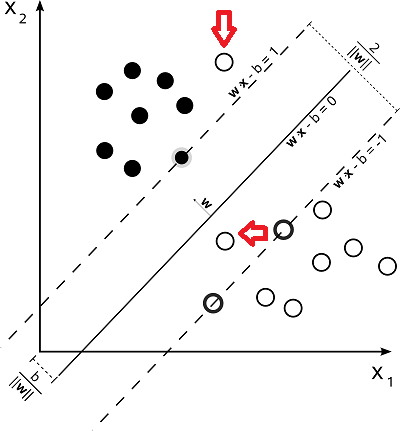
Próbki należące do zbioru, których lokalizacja wyznacza położenie marginesu separującego nazywane są wektorami nośnymi (ang. *support vectors*)[[12]](#footnote-15). Wizualizacja granicy decyzyjnej oraz marginesu separującego próbki dwóch klas została przedstawiona na Rys. 2.

Rys. 2 – Wyznaczanie hiperpłaszczyzny liniowego klasyfikatora SVM oraz maksymalizacja marginesu separującego. Źródło: A Review for Detecting Gene-Gene Interactions Using Machine Learning Methods in Genetic Epidemiology - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Linear-SVM-with-maximum-margin-hyperplane\_fig6\_258524366, dostęp z dnia 09.03.2024 r.



Istnieje modyfikacja algorytmu umożliwiająca rozwiązywanie za pomocą maszyny wektorów nośnych również problemów nieseparowalnych liniowo. Zawiera ona koncepcję tak zwanej zmiennej uzupełniającej (ang. *slack variable*) , 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.

Rys. 3 – Naruszenia miękkiego marginesu. Źródło: Lee (https://stats.stackexchange.com/users/98821/lee), SVM - non separable case/soft margin, where are the support vectors, URL (version: 2016-01-22): <https://stats.stackexchange.com/q/191928>, dostęp z dnia 09.03.2024 r.



Przekształcenie algorytmu polega na dodaniu zmiennej do zestawu ograniczeń *(13),* dzięki czemu do modelu zostaje wprowadzona większa elastyczność *(15).*

(15)

W tym przypadku, minimalizuje się wyrażenie nieco inne niż poprzednio, posiadające dodatkowy składnik (16). Parametr umożliwia kontrolowanie wysokości kary za nieprawidłowe przypisanie do klas. Im większa wartość parametru , tym bardziej rygorystyczny trening algorytmu. Można uznać tę modyfikację za rodzaj regularyzacji modelu.[[13]](#footnote-16)

(16)

[Regularyzacja L2 + Squared hinge loss?]

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. [[14]](#footnote-17)

### Wielomianowy naiwny klasyfikator bayesowski

Naiwne klasyfikatory bayesowskie 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 *(17).*

(17)

W powyższym wzorze oznacza zmienną celu, a to zależny wektor cech. Zapis określa natomiast prawdopodobieństwo warunkowe, czyli w tym wypadku prawdopodobieństwo wystąpienia pod warunkiem zajścia .

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 za pomocą wektorów . Oznaczenie symbolizuje liczbę cech, czyli w wypadku klasyfikacji tekstu – wielkość słownictwa. określa prawdopodobieństwo wystąpienia cechy w próbce należącej do klasy . Parametry są szacowane metodą liczenia częstości względnych *(18).*

(18)

W przedstawionym wzorze, wyraża ilość wystąpień cechy w próbkach klasy w zbiorze treningowym, natomiast jest całkowitą liczbą wystąpień wszystkich cech w próbkach klasy zbioru treningowego. Parametr wygładzania addytywnego został oznaczony literą .

### Drzewo decyzyjne

Model drzewa decyzyjnego 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 *(19)*. Najbardziej pożądanym rezultatem jest podział danych w węźle na jak najbardziej jednolite węzły potomne, wówczas funkcja ta przyjmuje maksymalne wartości.

(19)

W powyższym wzorze przyjęto oznaczenie jako wybraną miarę zanieczyszczenia, literę jako symbol liczby węzłów potomnych, a i jako zestaw danych odpowiednio węzła nadrzędnego i j-tego węzła potomnego. Analogicznie, symbole i określają całkowitą liczbę próbek w tych węzłach. Parametr definiuje charakterystykę, na podstawie której nastąpi rozgałęzianie. Przedstawione równanie najczęściej sprowadza się jednak do nieco prostszej formy *(20)*, ponieważ ze względu na mniejszą złożoność często stosuje się drzewo binarne, czyli takie, w którym węzeł rozgałęzia się maksymalnie na dwie podgrupy.

(20)

Istnieje kilka popularnych miar zanieczyszczeń. Jako przykład takiego kryterium może posłużyć wskaźnik Giniego *(21).* Uzyskuje on najwyższą wartość, gdy rozkład klas w węźle jest równomierny. Wyrażenie symbolizuje odsetek próbek klasy w węźle .

(21)

### Ekstremalne wzmocnienie gradientowe

Algorytm ekstremalnego gradientowego wzmocnienia drzew, bardziej znany jako XGBoost, został pierwotnie przedstawiony w 2016 roku[[15]](#footnote-20). 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 finalne predykcje będą wystarczająco precyzyjne. Wzmocnienie gradientowe (ang. *Gradient Boosting*) jest techniką, w której parametrami uczącymi przekazywanymi do następnika są błędy resztowe poprzednika, zwane też błędami rezydualnymi[[16]](#footnote-21).

Omawiając ideę działania ekstremalnego wzmocnienia gradientowego drzew dla problemu klasyfikacyjnego, najlepiej jest rozpocząć od przedstawienia funkcji celu *(22)* minimalizowanej podczas budowy danego drzewa.

(22)

Powyższe wyrażenie składa się z dwóch części. Pierwsza z nich to funkcja kosztu określona jako suma odstępstw predykcji modelu od rzeczywistej etykiety każdej z próbek, po niej natomiast następuje termin regularyzacji z parametrem . Symbol oznacza liczbę końcowych węzłów. Ponieważ składowa bierze udział w przycinaniu drzewa kiedy jest ono już w pełni zbudowane, na potrzeby objaśnień można chwilowo ją pominąć. to wartość wyjściowa rozpatrywanego liścia. Rozpatrując pierwsze drzewo w serii, wyrażenie można zastąpić sumą wartości początkowego przypuszczenia i wartości wyjściowej rozpatrywanego liścia tego drzewa. Poniżej przedstawiono wynikowe równanie *(23).*

(23)

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

Oszacowanie wartości samej funkcji kosztu można przeprowadzić korzystając z rozwinięcia Taylora[[17]](#footnote-22) drugiego rzędu, który jednocześnie sprowadza jej wzór do bardziej elementarnej formy *(24).* Według nazewnictwa stosowanego w definicji algorytmu XGBoost, pierwsza pochodna funkcji kosztu zostaje zastąpiona oznaczeniem symbolizującym gradient, a druga pochodna wyrażona jest literą jako reprezentacja hessiana.

(24)

Teraz można rozwinąć wzór *(23)* i podstawić wynik przekształcenia *(24)* w miejsce funkcji kosztu. Zabieg ten został przedstawiony w równaniu *(25).*

(25)

Warto zwrócić uwagę na to, że w powyższej formule znajdują się człony nie zawierające elementu . Przed przeszukiwaniem wartości tego komponentu w celu optymalizacji funkcji celu, można uprościć wzór eliminując wspomniane człony, ponieważ nie mają one wpływu na zmienną . Następnie należy uporządkować otrzymane wyrażenie *(26).* Warto podkreślić, że po uproszczeniu wzór *(25)* nie jest już równoważny formule *(26)*, ponieważ została z niego wyeliminowana pewna stała. Nie wpływa to jednak na wartość rozwiązania samego problemu optymalizacyjnego.

(26)

Wyłonienie optymalnej wartości oznacza znalezienie punktów będących rozwiązaniami równania różniczkowego *(27).*

(27)

Symbole oraz 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 *(28).*

(28)

Przed wyznaczeniem i , należy nałożyć funkcję logitową na prawdopodobieństwa *(29).*

(29)

Gradientem *(30)* nazywana jest pierwsza pochodna funkcji, hessianem *(31)* natomiast jej druga pochodna. Po wyznaczeniu pochodnych należy zastosować przekształcenie odwrotne do funkcji logitowej i tym samym wyeliminować wyrażenia zawierające funkcję eksponencjalną oraz logarytmy*.*

(30)

(31)

Kolejnym krokiem jest podstawienie otrzymanych wyrażeń do ostatecznej formy wzoru na *(27)*. Oznaczając rezyduum jako powstaje formuła *(32).*

(32)

Omawiając algorytm XGBoost należy również zdefiniować miarę podobieństwa . Wyraża się ją wzorem podobnym do wartości wyjściowej liścia, jednak licznik ułamka jest podniesiony do drugiej potęgi *(33).* Dzięki mierze podobieństwa, możliwe jest obliczenie zysku dla wybranego podziału węzła nadrzędnego na węzły potomne *(34)*. Umożliwia to określenie, która z dostępnych dla danego zbioru reguł jest najskuteczniejsza dla aktualnego stanu rozbudowy modelu.

(33)

(34)

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 *(35).* Następnie algorytm porównuje ją z wartością ustaloną domyślnie. Jeśli wartość domyślna jest wyższa od uzyskanej, liść nie zostanie utworzony.

(35)

Przycinanie wygenerowanego drzewa odbywa się na podstawie wyniku odejmowania ustalonego parametru od wartości najniższej istniejącej gałęzi. Jeśli opisana różnica jest ujemna, gałąź zostaje usunięta. Ten proces powtarzany jest iteracyjnie aż do napotkania gałęzi o pozytywnej wartości .

Schemat modelowania za pomocą algorytmu XGBoost rozpoczyna się od wyznaczenia wartości początkowego przypuszczenia i obliczenia na tej podstawie błędów rezydualnych dla wszystkich próbek zbioru treningowego. Ponadto, zdefiniowany zostaje parametr regularyzacji . Za pomocą miary podobieństwa oraz wartości zysku 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 indywidualnie dla każdego liścia w drzewie. Dla każdej próbki wartość wyjściowa liścia, w którym ona się znajduje, jest mnożona przez parametr uczenia , a do uzyskanego iloczynu dodawana jest funkcja logitowa z wartości początkowego przypuszczenia . Po konwersji wyników na prawdopodobieństwa z użyciem funkcji logistycznej, można zaktualizować wartości prawdopodobieństw 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 , które nie są już identyczne dla wszystkich przykładów. Wartości wyjściowe drugiego drzewa są również mnożone przez parametr i dodawane do wszystkich poprzednich predykcji. Szereg modeli zostaje zasilony o nowy element. Analogicznie do poprzednika, model definiuje nowe wartości . W ten sposób w algorytmie XGBoost budowane są kolejne drzewa aż do osiągnięcia warunku stopu.[[18]](#footnote-24)

## Sztuczne sieci neuronowe

Sztuczna sieć neuronowa (ang. *artificial neural network*) to koncepcja inspirowana działaniem ludzkiego układu nerwowego powstała pierwotnie w 1943 roku[[19]](#footnote-25). Najmniejszą jednostką sieci jest neuron, który na wzór swojego biologicznego odpowiednika ma za zadanie przesyłanie sygnału, informacji. Posiada on blok sumujący 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. [wzory funkcji aktywacji?] [uwaga! Softmax został opisany w sekcji regresji logistycznej]

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[[20]](#footnote-26). 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[[21]](#footnote-27) (ang. *layer normalization*) lub paczek danych[[22]](#footnote-28) (ang. *batch normalization*) w celu stabilizacji treningu. Z kolei blok porzucania (ang. *dropout*) jest jedną z najpopularniejszych technik regularyzacji sieci neuronowej obok regularyzacji L2 (ang. *ridge regression*) dążącej do zmniejszenia wag modelu. [uwaga! L2 została opisana w sekcji regresji logistycznej] 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[[23]](#footnote-29). 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[[24]](#footnote-30).

Warto dodać, że w przypadku złożonych problemów i stosowania głębokich sieci modele są narażone na wystąpienie problemu zanikających lub eksplodujących gradientów[[25]](#footnote-31), 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[[26]](#footnote-32) (ang. *adaptive moment estimation*), które są szybsze od tradycyjnej techniki gradientu prostego[[27]](#footnote-33).

### Rekurencyjne sieci neuronowe

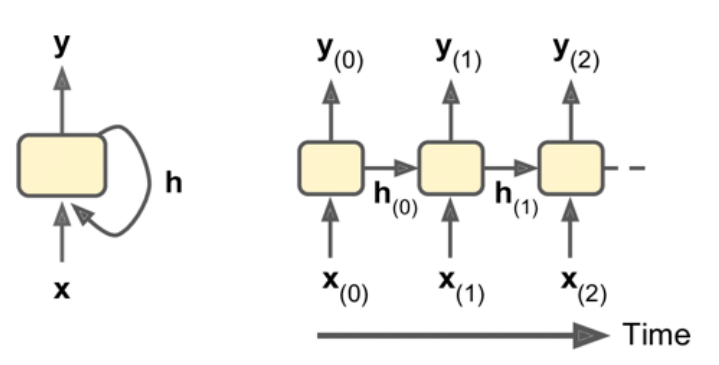
Sieci rekurencyjne (ang. *recurrent neural networks*) są odpowiednim wyborem dla danych sekwencyjnych, czyli takich, w których kolejność poszczególnych elementów próbki ma znaczenie. W tym przypadku, znaczące w modelowaniu jest uwzględnienie relacji między następującymi po sobie fragmentami ciągów. W tym celu powstała koncepcja neuronu rekurencyjnego. W każdym takcie , zwanym także krokiem czasowym (ang. *time step*) taki neuron oprócz danych wejściowych otrzymuje również własne wyniki z poprzedniego taktu , 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 dane wejściowe oraz swój poprzedni wynik , 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 , a dla danych wyjściowych z poprzedniego taktu znajdują się w tablicy . Wyniki opisanej warstwy dla pojedynczej próbki można wyrazić wzorem (36), gdzie oznacza funkcję aktywacji, najczęściej ReLU lub tangens hiperboliczny, a to obciążenia każdego neuronu.

(36)

Rozważając istotę przedstawionych struktur, można wywnioskować, że zawierają one coś na kształt pamięci, ponieważ wyjście w danym kroku czasowym 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 opisywany jest wzorem . Pojęcie to zostało wyodrębnione, ponieważ w bardziej skomplikowanych komórkach wyjście nie będzie równoważne z wartością jej stanu . 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 A., Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, 2018, s. 380



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[[28]](#footnote-34) 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 występująca zaraz obok stanu ukrytego , 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 A., Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, 2018, s. 398

Obraz zawierający tekst, diagram, Plan, linia

Opis wygenerowany automatycznie

W komórce LSTM wektor wejściowy i poprzedni wektor stanu są przekazywane do czterech różnych warstw. Główna warstwa występuje również w podstawowej komórce rekurencyjnej i zwraca wartość po przetworzeniu uzyskanych danych. Bramka zapominająca (ang. *forget gate*) decyduje, które informacje w pamięci długotrwałej zostaną usunięte, jej wyjście stanowi wektor . Bramka wejściowa (ang. *input gate*) przyczynia się do częściowego zachowania informacji zwracanych przez warstwę główną poprzez selekcję elementów wyjścia , które należy dodać do pamięci długoterminowej. Jej wektor wynikowy oznaczany jest symbolem . Bramka wyjściowa (ang. *output gate*) definiuje składniki stanu długoterminowego, które są brane pod uwagę przy generowaniu nowych wektorów i w danym takcie . Jej wyjście opisuje się jako . Warto dodać, że warstwa główna jako jedyna używa tangensa hiperbolicznego w roli funkcji aktywacji. Pozostałe trzy bramki korzystają z funkcji logistycznej. Równania *(37)*, *(38)*, *(39)* i *(40)* opisują operacje wykonywane przez poszczególne warstwy w kroku obliczeniowym dla pojedynczej próbki.

(37)

(38)

(39)

(40)

Wygenerowanie nowego stanu pamięci długoterminowej polega na przepuszczeniu poprzedniego stanu przez bramkę zapominającą, gdzie usuwana jest część jego zawartości, a następnie na zsumowaniu wynikowego wektora z nowymi znaczącymi informacjami wybranymi przez bramkę wejściową. Zdefiniowanie wartości , polega natomiast na obliczeniu tangensa hiperbolicznego dla kopii otrzymanego stanu , a rezultat tego działania jest filtrowany przez wytyczne wygenerowane przez bramkę wyjściową. Wartość w tym przypadku jest równa . Obliczenia stanów pamięci dla pojedynczej próbki zostały opisane równaniami *(41)* i *(42)*.

(41)

(42)

W 2014 roku opublikowano uproszczony wariant komórki LSTM osiągający jednak podobne rezultaty. Jednostka GRU[[29]](#footnote-35) odznacza się wprowadzeniem strategii współdziałania bramki zapominającej i wejściowej, a także używaniem tylko jednego stanu pamięci , 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 A., Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, 2018, s. 400

Obraz zawierający tekst, diagram, linia, zrzut ekranu

Opis wygenerowany automatycznie

Poniżej przedstawiono wzory opisujące działanie poszczególnych warstw dla pojedynczego przykładu w takcie *(43)(44)(45)*. Formuła *(46)* określa w jaki sposób uzyskuje się wartości nowego stanu pamięci.[[30]](#footnote-36)

(43)

(44)

(45)

(46)

## Mechanizm Attention

W 2017 roku, w szeroko później cytowanym artykule pod tytułem „*Attention Is All You Need*”[[31]](#footnote-37), 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. [Softmax? Wzory? Self-Attention?]

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: Ashish Vaswani i in., Attention Is All You Need”, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2023

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

## Trening i ewaluacja modeli

### 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 -krotna, która polega na rozbiciu zbioru danych na podzbiorów, a następnie przeprowadzeniu treningów wybranej architektury, za każdym razem traktując jako zbiór testowy inną spośród stworzonych grup. Ogólną jakość modelu można ocenić uśredniając wyniki takich klasyfikacji. Jako kryterium oceny skuteczności modelu zazwyczaj przyjmuje się miarę dokładności.

### 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.

### Funkcja kosztu – entropia krzyżowa

Z dodatkiem funkcji Softmax wyjaśniona w Równaniu 6 przy okazji regresji logistycznej – przenieść tu?

### 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.

[Wskaźnik F1]

# Materiały i metody



## 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[[32]](#footnote-38)* oraz *TweetTokenizer[[33]](#footnote-39)* , 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 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', '.' |

## 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 [[34]](#footnote-40). 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', '!' |
| 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[[35]](#footnote-41), 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.

[Tabela średnich liczności lub wykres]

## 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[[36]](#footnote-42)*. 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.

## Klasyfikacja za pomocą tradycyjnych modeli uczenia maszynowego

### 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[[37]](#footnote-43)*. Parametr *max\_features* definiujący rozważaną wielkość słownika ustawiono na wartość 5000. Wynik tej modyfikacji stanowił wejście do funkcji *TfidfTransformer[[38]](#footnote-44)* będącej cyfrową reprezentacją modyfikacji   
TF-IDF. 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.

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

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

### Dobór modeli

Pierwszym z zastosowanych algorytmów była wielomianowa regresja logistyczna zaimplementowana przy pomocy funkcji *LogisticRegression[[39]](#footnote-45)* z biblioteki *scikit-learn.* W fazie eksperymentów zauważono wzrost wartości miary dokładności wraz ze zwiększaniem parametru *C* (Rys. 8), dlatego dobrano dla niego wartość 1. Regularyzacja jest wbudowana w metodę i wykonywana domyślnie, w przypadku tego problemu była to regularyzacja L2. Podczas treningu jako funkcję kosztu przyjęto entropię krzyżową.

Rys. 8 – Wzrost dokładności regresji logistycznej na zbiorze testowym przy wzroście parametru C. Źródło: opracowanie własne

Obraz zawierający tekst, linia, zrzut ekranu, Wykres

Opis wygenerowany automatycznie

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[[40]](#footnote-46)*. Podobnie jak w przypadku regresji liniowej, przeprowadzono analizę parametru C, jednak na Rys. 9 można zauważyć, że tendencja w tym wypadku nie była monotoniczna. Jako optymalną wartość wybrano 0.4. Ponadto, jako funkcję kosztu wybrano Squared Hinge, która dobrze sprawdza się w treningu maszyn wektorów nośnych w zadaniu optymalizacyjnym znajdywania maksymalnego marginesu. Podczas treningu stosowano regularyzację L2.

Rys. 9 – Dokładność klasyfikatora SVM na zbiorze testowym przy wzroście parametru C. Źródło: opracowanie własne

Obraz zawierający tekst, diagram, Wykres, linia

Opis wygenerowany automatycznie

Następnym tradycyjnym modelem uczenia maszynowego wybranym do klasyfikacji szesnastu typów osobowości był wielomianowy naiwny klasyfikator bayesowski w implementacji *MultinomialNB*[[41]](#footnote-47). Wzrost parametru wygładzania addytywnego był jednoznaczny ze zmniejszaniem się dokładności modelu, dlatego ustawiono jego wartość na niewielką, równą 0.1. Zaobserwowane zjawisko przedstawiono na Rys. 10.

Rys. 10 – Spadek dokładności wielomianowego naiwnego klasyfikatora bayesowskiego na zbiorze testowym przy wzroście parametru alpha. Źródło: opracowanie własne

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Ostatnim zaproponowanym algorytmem było ekstremalne wzmocnienie gradientowe. W tym celu wytrenowano klasyfikator *XGBClassifier* wywodzący się z modułu *XGBoost[[42]](#footnote-48)*. 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 oraz lambdy związanej z regularyzacją L2 na zbiorze zakończyło się wyłonieniem najlepszej pary wartości – odpowiednio liczby 6 i 1.

## Modelowanie z zastosowaniem głębokich sieci neuronowych

### Kodowanie danych

Przygotowanie reprezentacji numerycznej tekstu dla modeli uczenia głębokiego rozpoczęto od stworzenia słownika zawierającego wyrazy występujące we wstępnie przygotowanych i oczyszczonych postach zbioru treningowego. Spośród ponad 108 000 słów wykrytych przez funkcję *Tokenizer[[43]](#footnote-49),*  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. 11). Po przygotowaniu opisanej struktury, przeprowadzono na jej podstawie proste mapowanie próbek do reprezentacji liczbowej (Rys. 12).

Rys. 11 – Pierwsze rekordy słownika z użyciem metody fit\_on\_texts klasy Tokenizer. Źródło: opracowanie własne

Obraz zawierający tekst, paragon, Czcionka, biały

Opis wygenerowany automatycznie

Rys. 12 – Przykładowe fragmenty próbek i ich reprezentacji numerycznej. Źródło: opracowanie własne

Obraz zawierający tekst, Czcionka, zrzut ekranu, biały

Opis wygenerowany automatycznie

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[[44]](#footnote-50)*.

### 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*[[45]](#footnote-51)osadzającej słowa w przestrzeni 64-wymiarowej, uwzględniając ich podobieństwa znaczeniowe. Ten segment można traktować jako dopełnienie dotychczasowych działań w kontekście tworzenia funkcjonalnej reprezentacji numerycznej próbek. Kolejnym komponentem modelu była pojedyncza komórka *GRU*[[46]](#footnote-52) zawierająca 512 jednostek wraz z warstwą normalizującą *BatchNormalization*[[47]](#footnote-53). Po niej wystąpiła warstwa gęsta *Dense[[48]](#footnote-54)* złożona z 128 neuronów z funkcją aktywacyjną ReLU, następnie mechanizm porzucania, czyli segment *Dropout[[49]](#footnote-55)* odrzucający losowo30% 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. 13.

Rys. 13 – Schemat architektury rekurencyjnej sieci neuronowej z warstwą GRU. Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

Entropia krzyżowa w wariancie dla klasyfikacji wieloklasowej odgrywała rolę funkcji kosztu podczas treningu, a algorytm *Adam*[[50]](#footnote-56) o domyślnych parametrach został wybrany do optymalizacji. Jako metrykę oceny skuteczności modelu obrano implementację dokładności *CategoricalAccuracy[[51]](#footnote-57)*, oraz *F1Score[[52]](#footnote-58)* z uśrednianiem *macro*.

[dodać wersję z KerasTuner?]

### 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[[53]](#footnote-59)*. 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*[[54]](#footnote-60), a następnie procedura *GlobalAveragePooling1D[[55]](#footnote-61)* 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[[56]](#footnote-62) o współczynniku 0.01. Na zakończenie dobudowano mechanizm porzucania, pomijający losowo 30% neuronów i gęstą warstwę wyjściową z aktywacją Softmax. Rys. 14 ilustruje schemat architektury zaproponowanego modelu.

Rys. 14 – Schemat architektury sieci neuronowej z mechanizmem Attention. Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

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.

[BERT?]

## 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[[57]](#footnote-63)*.

W procesie przygotowania danych fundamentalną rolę odegrał moduł przeznaczony do analizy i manipulacji danymi *pandas* 2.1[[58]](#footnote-64) oraz biblioteka do operacji naukowych *NumPy* 1.26[[59]](#footnote-65). Budowanie wykresów opierało się natomiast na interfejsie *matplotlib.pyplot[[60]](#footnote-66)*. Do implementacji klasycznych modeli uczenia maszynowego posłużyły głównie narzędzia analityczne modułu *scikit-learn[[61]](#footnote-67).* Sieci neuronowe były inicjowane i trenowane przy pomocy interfejsu *Keras[[62]](#footnote-68)* biblioteki *Tensorflow* 2.10.

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.

# Wyniki



## Wieloklasowość

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

Popularnym w literaturze 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 (47). Jeśli dla każdej próbki zbioru testowego etykieta zostałaby wylosowana, przy czym prawdopodobieństwo przypisania każdej z klas byłoby takie samo, przy dodatkowym założeniu, że zbiór testowy był wystarczająco duży, procent odpowiednio zaklasyfikowanych próbek oscylowałby właśnie wokół tego poziomu.

(47)

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.

## 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 F1 w wariancie *macro,* zwracającym średnią wartości F1 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 6 – 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 | |
| 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%** |

W Tabeli 6 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 6 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 7 – Porównanie wartości wskaźnika F1 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 | |
| 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 7 zaprezentowano uzyskane podczas testów uśrednione wartości wskaźnika F1 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 F1 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 poprzednim rozdziale, 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. Sieć neuronowa z modułem Attention uczyła się przez 20 epok, jednorazowo przetwarzając ośmioelementową podgrupę próbek. Na Rys. 15 i Rys. 16 przedstawiono ewaluację modeli podczas trwania treningu za pomocą metryki dokładności.

Rys. 15 – Wykres dokładności sieci rekurencyjnej z komórką GRU. Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, linia, Wykres

Opis wygenerowany automatycznie

Rys. 16 – Wykres dokładności sieci neuronowej z modułem Attention. Źródło: opracowanie własne

Obraz zawierający tekst, Wykres, linia, zrzut ekranu

Opis wygenerowany automatycznie

Tabela 8 przedstawia porównanie wyników inferencji zbioru testowego i treningowego z użyciem opisanych modeli uczenia głębokiego. Model z warstwą GRU uzyskał lepsze wyniki niż sieć z mechanizmem Attention, jednak nie poradziły sobie z zadaniem tak dobrze jak niektóre tradycyjne modele uczenia maszynowego w opisanym wcześniej ujęciu.

Tabela 8 – 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 F1 | **32.67%** | **99.48%** |
| Wskaźnik F1  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%  0%  29.41%  0%  0%  0% | 99.91%  99.55%  98.63%  99.89%  100%  100%  99.45%  99.72%  99.54%  99.81%  99.62%  96.20%  99.30%  100%  100%  100% |
| Sieć z modułem Attention | Dokładność | 48.99% | 97.58% |
| Wskaźnik F1 | 26.71% | 76.32% |
| Wskaźnik F1 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 F1 zostały uzyskane poprzez uśrednienie jego wartości obliczonych osobno dla każdej z szesnastu klas – przykładowy ich rozkład można zaobserwować w Tabeli 8. 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 F1 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.

[Ewentualne porównanie do innych architektur? Czasy treningów?]

# Podsumowanie



W ramach tej pracy przeprowadzono badania na temat klasyfikacji typów osobowości modelu MBTI na podstawie wpisów użytkowników z portalu społecznościowego.

[Co po kolei robiono?]

Wśród przeprowadzonych eksperymentów wyłoniono algorytm ekstremalnego wzmocnienia gradientowego uruchomiony na danych przygotowanych techniką TF-IDF na bazie unigramów jako najskuteczniejsza z wszystkich przetestowanych metod rozwiązania postawionego problemu badawczego. Wspomniany model osiągnął wartość dokładności na poziomie 69.16% na zbiorze testowym przy wskaźniku F1 przyjmującym wartość 56.49%. Uzyskany wynik stanowi obiecujące rozwiązanie dla zadania klasyfikacji typów osobowości na podstawie analizy postów internetowych. Problemy natury psychologicznej poddane analizie danych mogą charakteryzować się większymi nieścisłościami predykcji ze względu na różnorodność grupy badawczej i ludzką złożoność.

Stworzony model mógłby posłużyć jako wsparcie pracy rekruterów biznesowych czy jako silnik poręcznego narzędzia podpiętego do portali społecznościowych i wspomagającego rozwój osobisty jego użytkowników.

[Cel pracy został osiągnięty. + Rozważania nt. efektów w kontekście osobowości. + Możliwe rozwinięcia projektu.]

Input

Embedding

MultiHead Attention

Layer Normalization

Global Average Pooling 1D

Dense

Dropout

Dense

Output

# Roboczy generator schematów

Co się usunęło: część odniesień do bibliografii ☹ + opisać obrazki/tabele bibliografią (dodać ew. komentarze)

# Spis tabel

Tabela 1 – Opis par kategorii modelu MBTI. Źródło: opracowanie własne 11

Tabela 2 – Porównanie tokenizacji tekstu przez word\_tokenizer i TweetTokenizer. Źródło: opracowanie własne 43

Tabela 3 – Proces wstępnej obróbki tekstu. Źródło: opracowanie własne 43

Tabela 4 – Przykłady oznaczania charakterystycznych elementów w próbkach. Źródło: opracowanie własne 44

Tabela 5 – Przykładowe rekordy słownika w zależności od numeru próby. Źródło: opracowanie własne 47

Tabela 6 – Porównanie dokładności klasyfikacji klasycznych modeli uczenia maszynowego odpowiednio dla zbioru testowego i treningowego w zależności od strategii tworzenia worka słów. Źródło: opracowanie własne 56

Tabela 7 – Porównanie wartości wskaźnika F1 klasycznych modeli uczenia maszynowego dla klasyfikacji odpowiednio zbioru testowego i treningowego w zależności od strategii tworzenia worka słów. Źródło: opracowanie własne 57

Tabela 8 – Porównanie wskaźników oceny zastosowanych modeli opartych na sieciach neuronowych. Źródło: opracowanie własne 59

# Spis rysunków

Rys. 1 – Dystrybucja próbek między klasami. Źródło: opracowanie własne 13

Rys. 2 – Wyznaczanie hiperpłaszczyzny liniowego klasyfikatora SVM oraz maksymalizacja marginesu separującego. Źródło: A Review for Detecting Gene-Gene Interactions Using Machine Learning Methods in Genetic Epidemiology - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Linear-SVM-with-maximum-margin-hyperplane\_fig6\_258524366, dostęp z dnia 09.03.2024 r. 23

Rys. 3 – Naruszenia miękkiego marginesu. Źródło: Lee (https://stats.stackexchange.com/users/98821/lee), SVM - non separable case/soft margin, where are the support vectors, URL (version: 2016-01-22): https://stats.stackexchange.com/q/191928, dostęp z dnia 09.03.2024 r. 24

Rys. 4 – Komórka rekurencyjna i jej stan ukryty (po lewej) oraz jej rozwijanie w czasie (po prawej). Źródło: Géron A., Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, 2018, s. 380 35

Rys. 5 – Struktura komórki LSTM. Źródło: Géron A., Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, 2018, s. 398 36

Rys. 6 – Struktura komórki GRU. Źródło: Géron A., Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, 2018, s. 400 38

Rys. 7 – Wizualizacja architektur mechanizmu Attention. Źródło: Ashish Vaswani i in., Attention Is All You Need”, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2023 39

Rys. 8 – Wzrost dokładności regresji logistycznej na zbiorze testowym przy wzroście parametru C. Źródło: opracowanie własne 48

Rys. 9 – Dokładność klasyfikatora SVM na zbiorze testowym przy wzroście parametru C. Źródło: opracowanie własne 49

Rys. 10 – Spadek dokładności wielomianowego naiwnego klasyfikatora bayesowskiego na zbiorze testowym przy wzroście parametru alpha. Źródło: opracowanie własne 49

Rys. 11 – Pierwsze rekordy słownika z użyciem metody fit\_on\_texts klasy Tokenizer. Źródło: opracowanie własne 50

Rys. 12 – Przykładowe fragmenty próbek i ich reprezentacji numerycznej. Źródło: opracowanie własne 51

Rys. 13 – Schemat architektury rekurencyjnej sieci neuronowej z warstwą GRU. Źródło: opracowanie własne 52

Rys. 14 – Schemat architektury sieci neuronowej z mechanizmem Attention. Źródło: opracowanie własne 53

Rys. 15 – Wykres dokładności sieci rekurencyjnej z komórką GRU. Źródło: opracowanie własne 58

Rys. 16 – Wykres dokładności sieci neuronowej z modułem Attention. Źródło: opracowanie własne 59

# Wykaz symboli i oznaczeń

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

* a (mała litera kursywą) oznacza skalar,
* **a** (mała pogrubiona litera) oznacza wektor]

# Wykaz używanych skrótów



* MBTI – Myers-Briggs Type Indicator®

# 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

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.

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

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

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

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 [↑](#footnote-ref-1)
2. <https://www.recruiter.com/recruiting/critique-of-the-myers-briggs-type-indicator-critique/>, dostęp z dnia 24.11.2023 r. [↑](#footnote-ref-2)
3. <https://www.wired.com/story/myers-briggs-test-internet-fans/>, dostęp z dnia 24.11.2023 r. [↑](#footnote-ref-3)
4. <https://www.reddit.com/> [↑](#footnote-ref-4)
5. <https://www.16personalities.com/pl> [↑](#footnote-ref-5)
6. <https://www.theguardian.com/science/brain-flapping/2013/mar/19/myers-briggs-test-unscientific>, dostęp z dnia 24.11.2023 r. [↑](#footnote-ref-6)
7. Na podstawie raportu <https://eu.themyersbriggs.com/-/media/Files/PDFs/Technical-information/MBTI_reliability_and_validity_info.pdf> [↑](#footnote-ref-7)
8. Charakterystyki w *Tabeli 1* są skrótem bardziej szczegółowych opisów przedstawionych w artykule <https://potencjalosobowosci.com/mbti/>, dostęp z dnia 01.02.2024 r. [↑](#footnote-ref-8)
9. <https://www.kaggle.com/datasets/datasnaek/mbti-type> [↑](#footnote-ref-9)
10. <https://www.personalitycafe.com/> [↑](#footnote-ref-10)
11. <https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce>, dostęp z dnia 21.02.2024 r. [↑](#footnote-ref-11)
12. Raschka S., Mirjalili V., *Python. Uczenie maszynowe. Wydanie II,* Helion, 2019, s. 88-90 [↑](#footnote-ref-15)
13. Raschka S., Mirjalili V., *Python. Uczenie maszynowe. Wydanie II,* Helion, 2019, s. 90-92 [↑](#footnote-ref-16)
14. Steven S. Skiena, *The Data Science Design Manual*, Edition 1, Springer Cham, 2017, s. 366-369 [↑](#footnote-ref-17)
15. Chen, T., Guestrin, C., *XGBoost: A Scalable Tree Boosting System*, in Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, California, San Francisco, USA, August 13 - 17, 2016, 785–794 [↑](#footnote-ref-20)
16. <https://www.geeksforgeeks.org/xgboost/>, dostęp z dnia 10.03.2024 r. [↑](#footnote-ref-21)
17. <https://mathspace.pl/matematyka/genialny-wzor-taylora-czyli-o-informacji-zakodowanej-w-pochodnych/>, dostęp z dnia 10.03.2024 r. [↑](#footnote-ref-22)
18. 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. [↑](#footnote-ref-24)
19. McCulloch W. S., Pitts W., *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics 5, 115–133, 1943 [↑](#footnote-ref-25)
20. <https://www.sztucznainteligencja.org.pl/kurs/sztuczna-inteligencja-dla-poczatkujacych/sztuczne-sieci-neuronowe/>, dostęp z dnia 12.03.2024 r. [↑](#footnote-ref-26)
21. Ba J.L. i in., *Layer Normalization*, 2016 [↑](#footnote-ref-27)
22. Ioffe S., Szegedy Ch., *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015 [↑](#footnote-ref-28)
23. Rumelhart D. E. i in., *Learning Internal Representations by Error Propagation*, 1986 [↑](#footnote-ref-29)
24. Géron A., *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow,* Helion, 2018, s. 262-263 [↑](#footnote-ref-30)
25. Bengio Y. i in., *Learning long-term dependencies with gradient descent is difficult*, 1994 [↑](#footnote-ref-31)
26. Kingma D. P., Ba J., *Adam: A Method for Stochastic Optimization*, 2014 [↑](#footnote-ref-32)
27. Géron A., *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow,* Helion, 2018, s. 275-302 [↑](#footnote-ref-33)
28. Hochreiter S., Schmidhuber J., *Long Short-term Memory*, Neural computation 9, 1997 [↑](#footnote-ref-34)
29. Cho K. i in., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014 [↑](#footnote-ref-35)
30. Treść tego podrozdziału opiera się na książce Géron A., *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow,* Helion, 2018, s. 377-401 [↑](#footnote-ref-36)
31. Vaswani A. i in., *Attention Is All You Need,* 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2023 [↑](#footnote-ref-37)
32. <https://www.nltk.org/api/nltk.tokenize.word_tokenize.html> [↑](#footnote-ref-38)
33. <https://tedboy.github.io/nlps/generated/generated/nltk.tokenize.TweetTokenizer.html> [↑](#footnote-ref-39)
34. <https://sjp.pl/hashtag>, dostęp z dnia 18.11.2023 r. [↑](#footnote-ref-40)
35. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html> [↑](#footnote-ref-41)
36. <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html> [↑](#footnote-ref-42)
37. <https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html> [↑](#footnote-ref-43)
38. <https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html> [↑](#footnote-ref-44)
39. <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html> [↑](#footnote-ref-45)
40. <https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html> [↑](#footnote-ref-46)
41. <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html> [↑](#footnote-ref-47)
42. <https://xgboost.readthedocs.io/en/stable/get_started.html> [↑](#footnote-ref-48)
43. <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer> [↑](#footnote-ref-49)
44. <https://www.tensorflow.org/api_docs/python/tf/keras/utils/pad_sequences> [↑](#footnote-ref-50)
45. <https://keras.io/api/layers/core_layers/embedding/> [↑](#footnote-ref-51)
46. <https://keras.io/api/layers/recurrent_layers/gru/> [↑](#footnote-ref-52)
47. <https://keras.io/api/layers/normalization_layers/batch_normalization/> [↑](#footnote-ref-53)
48. <https://keras.io/api/layers/core_layers/dense/> [↑](#footnote-ref-54)
49. <https://keras.io/api/layers/regularization_layers/dropout/> [↑](#footnote-ref-55)
50. <https://keras.io/api/optimizers/adam/> [↑](#footnote-ref-56)
51. <https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalAccuracy> [↑](#footnote-ref-57)
52. <https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/F1Score> [↑](#footnote-ref-58)
53. <https://keras.io/api/layers/attention_layers/multi_head_attention/> [↑](#footnote-ref-59)
54. <https://keras.io/api/layers/normalization_layers/layer_normalization/> [↑](#footnote-ref-60)
55. <https://keras.io/api/layers/pooling_layers/global_average_pooling1d/> [↑](#footnote-ref-61)
56. [https://keras.io/api/layers/regularizers/ - l2-class](https://keras.io/api/layers/regularizers/#l2-class) [↑](#footnote-ref-62)
57. <https://www.anaconda.com/> [↑](#footnote-ref-63)
58. <https://pandas.pydata.org/pandas-docs/version/2.1/index.html> [↑](#footnote-ref-64)
59. <https://numpy.org/doc/stable/index.html> [↑](#footnote-ref-65)
60. [matplotlib.pyplot — Matplotlib 3.5.3 documentation](https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html) [↑](#footnote-ref-66)
61. <https://scikit-learn.org/stable/> [↑](#footnote-ref-67)
62. <https://keras.io/2.15/api/> [↑](#footnote-ref-68)