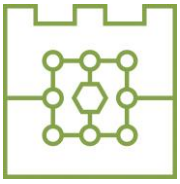
Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Informatyki i Telekomunikacji



Sebastian Sitko

Numer albumu:141011

Wyszukiwanie podobieństw semantycznych formularzy zgłoszenia błędów testowania programów

Searching for semantic similarities between program bug report forms

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem:
dr Radosław Kycia

Recenzent pracy:
dr Joanna Płażek

Kraków 2022

Streszczenie

W pracy opisano wyszukiwanie duplikatów formularzy zgłaszanych przez testerów błędów oprogramowania na podstawie danych pochodzących z organizacji Nokia Mobile Networks zajmującej się tworzeniem sprzętu 5G. W celach wyłonienia najlepszego potoku przetwarzania danych wykonano badania przy użyciu następujących metod: wektoryzacji dokumentów tekstowych z wykorzystaniem TFiDF oraz BERT; do wyszukiwania duplikatów użyto algorytmów PCA, LSA, t-SNE, UMAP, lasu losowego oraz głębokiej sieci o architekturze syjamskiej z funkcją potrójnej straty; do porównania dystansów wektorów próbek wykorzystano dystans cosinusowy oraz euklidesowy. Najlepsze wyniki uzyskano przy użyciu sieci syjamskiej.

Abstract

The paper describes the search of duplicate software bug forms reported by testers based on data from Nokia Mobile Networks' 5G hardware development organization. For the purpose of selecting the best data processing pipeline, tests were performed using the following methods: vectorization of text documents using TFiDF and BERT; PCA, LSA, t-SNE, UMAP, random forest and Siamese deep network with triple loss function algorithms were used to find duplicates; cosine distance and Euclidean distance were used to compare the distances of sample vectors. The best results were obtained using the Siamese network.

Spis treści

1 Wstęp **1**

1.1 Cel pracy . . . 1

1.2 Zakres pracy . . . 1

1.3 Metodyka pracy . . . 1

I Część teoretyczna **3**

2 Podstawy teoretyczne **4**

2.1 Metody przetwarzania danych kategorycznych . . . 4

2.2 Metody przetwarzania danych tekstowych . . . 4

2.2.1 Preprocesowanie tekstu . . . 5

2.2.2 TFiDF . . . 7

2.2.3 Koder - dekoder . . . 8

2.2.4 Warstwy atencji . . . 8

2.2.5 BERT . . . 10

2.3 PCA . . . 11

2.4 Truncated-SVD . . . 13

2.5 t-SNE . . . 13

2.6 UMAP . . . 15

2.7 Las losowy . . . 17

2.8 Modele głębokie uczenia maszynowego . . . 18

2.8.1 Model syjamski . . . 19

2.8.2 Funkcja potrójnej straty . . . 20

2.9 Metody porównywania podobieństwa wektorów . . . 21

2.9.1 Odległość Euklidesowa . . . 21

2.9.2 Podobieństwo cosinusowe i odległość cosinusowa . . . 21

II Część praktyczna **22**

3 Implementacja **23**

3.1 Analiza problemu . . . 23

3.2 Przebieg badań 23
3.3 Opis zbioru danych 24
3.4 Opis przetwarzania zbioru danych 24
3.4.1 TFiDF 26
3.4.2 BERT 26
3.5 Walidacja modeli 28
3.6 Klastrowanie danych z użyciem PCA 29
3.6.1 Analiza pól formularza z wykorzystaniem PCA oraz podobieństwa cosinusowego . 29
3.6.2 Wizualizacja danych z analizy PCA 30
3.6.3 Walidacja 30
3.7 Klastrowanie danych z użyciem Truncated-SVD 30
3.8 Klastrowanie danych z użyciem t-SNE 33
3.9 Klastrowanie danych z użyciem UMAP 34
3.10 Las losowy 35
3.11 Model syjamski z funkcją potrójnej straty 37
3.12 Walidacja 39
3.13 Dyskusja 40

4 Wnioski **42**

Literatura **43**

Rozdział 1

Wstęp

1.1 Cel pracy

W procesie testowania dedykowane zespoły odpowiadające za dany komponent szukają błędów i wypełniają formularz w przeznaczonym do tego narzędziu, gdy błąd zostanie znaleziony. Celem pracy było rozwiązanie praktycznego problemu występowania większej liczby zgłoszeń, niż rzeczywistych błędów w formularzach zgłoszenia przesyłanych przez testerów.

Występowanie duplikatów powoduje logistyczne problemy, a także zwiększone koszty naprawy. Jeśli we wczesnych etapach nie zostanie wykryte takie zjawisko, zespoły deweloperskie równolegle pracują nad naprawą tego samego. To niepożądane zjawisko jest rozpatrywane przez desygnowane do tego osoby, lecz ludzki umysł nie jest stworzony do analizy i zapamiętania tak dużej ilości zgłoszeń, by porównać ich zawartość. Rozwiązaniem jest stworzenie narzędzia, które mogłoby usprawnić ten proces. Praca odpowiada na pytanie, czy użycie sztucznej inteligencji pozwoli na szybsze i skuteczniejsze rozpoznawania duplikatów. Cel był motywowany rzeczywistym problemem w firmie Nokia.

1.2 Zakres pracy

W zakres pracy wchodzi analiza zbioru danych formularzy zgłoszenia błędów przez testerów oprogramowania i opracowanie modelu uczenia maszynowego odpowiedniego do rozpoznawania duplikatów występujących w zbiorze formularzy. W pracy zrobiono przegląd metod sztucznej inteligencji i uczenia maszynowego, które mogą służyć do osiągnięcia celu. Ograniczono się do następujących metod: algorytmy redukcji wymiarowości zarówno liniowe (PCA, Truncated-SVD), jak i nieliniowe(t-SNE, UMAP), las losowy oraz bardziej zaawansowane architektury głębokiego uczenia maszynowego, których zastosowaniem jest grupowanie semantycznie bliskich sobie próbek.

1.3 Metodyka pracy

Badania prowadzone celem opracowania odpowiedniego sposobu rozpoznawania duplikatów podzielone były na trzy części. Początkowy pozwolił na analizę zbioru danych, wybranie odpowiednich parametrów i zrozumienie problemu. Drugim etapem było przetworzenie próbek danych, wyekstrahowanie jak naj-

większej ilości informacji i zapisanie ich w odpowiedniej formie. Ostatni etap polegał na odnalezieniu odpowiedniego podejścia uczenia maszynowego do rozwiązania postawionego problemu.

Zestaw narzędzia, które użyto w tym procesie - nazywany często *PyData Stack*, to biblioteki języka Python: *numpy*, *pandas*, *matplitlib* wraz z narzędziem *Jupyter Notebook* przeznaczone do wykorzystania w problemach z przetwarzaniem danych. Do implementacji modeli głębokich sztucznej inteligencji użyto biblioteki *Tensorflow*, natomiast implementacje algorytmów do redukcji wymiarowości pochodziły z bibliotek *sci-kit learn* oraz *umap*.

Wszystkie badania były przeprowadzone w kooperacji z Nokia Kraków na serwerze z pojedynczą jednostką procesora graficznego NVIDIA GeForce RTX 2070 SUPER o pamięci wewnętrznej 8GB.

Praca podzielona jest na dwie części. W części teoretycznej opisane są wykorzystywane algorytmy orz architektury sieci głębokich. Dodatkowo znajduje się tam także opis metod przetwarzania danych oraz wektoryzacji tekstu, wraz z metodami porównywania podobieństw wektorów. W części praktycznej analizowany jest problem duplikatów za pomocą wizualizacji danych, klasteryzacji na komponenty sprzętowe i oprogramowania oraz wyszukiwanie duplikatów w zbiorze formularzy.

Część I

Część teoretyczna

Rozdział 2

Podstawy teoretyczne

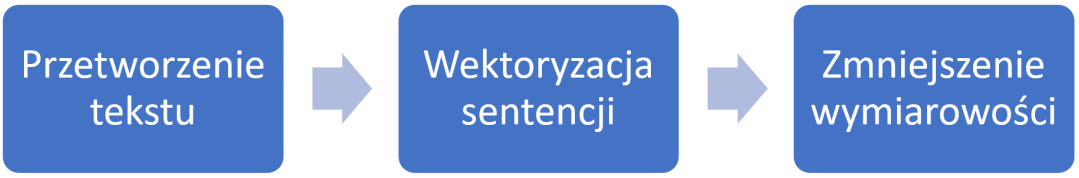
2.1 Metody przetwarzania danych kategorycznych

Dane kategoryczne [16] to rodzaj danych, które można podzielić na skończoną liczbę rozłącznych grup, tym samym przypisując im etykietę charakterystyczną dla elementu przynależnego do tejże kategorii. Analizując zbiór, możliwość klasteryzacji danych (podział na kategorie) pozwala na znajdowanie zależności między nimi tym samym zrozumienie struktury zbioru. Przykładem może być rodzaj zwierzęcia domowego. Przypisujemy go do jednej kategorii, nie może to być zarówno pies i kot w tym samym momencie. Psy i koty różnią się od siebie w sferze wizualnej. Dzieląc je na kategorie można zauważyć części wspólne dla poszczególnych klas, które pominięto by przy analizie wszystkich zwierząt. Kategoryzację danych można wykorzystać w dwojaki sposób w modelach uczenia maszynowego. Jako dane wejściowe reprezentują jeden z parametrów próbki określający przynależność do kategorii. W modelach klasyfikacji, dane wyjściowe są przyporządkowane kategoriom, model zwróci prawdopodobieństwo przynależności do każdej z kategorii. Tam gdzie znajdzie się najwyższa wartość, odpowiadająca jej grupa jest uznawana przez model za najbardziej prawdopodobny. W przeprowadzaniu badań przeprowadzono próbę wyróżnienia kategorii, do których należą formularze, by następnie użyć ich do klasyfikacji nowych wiadomości. Najczęstszym kodowaniem wartości kategorycznych jest kodowanie gorąco-jedynkowe, szerzej opisane w [7].

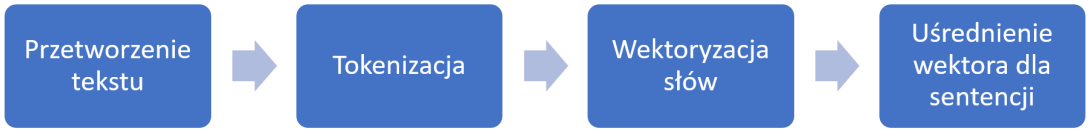
2.2 Metody przetwarzania danych tekstowych

Dane tekstowe nie są proste do reprezentowania dla algorytmów uczenia maszynowego, ponieważ liczba wariacji znaków w wyrazach jest bardzo duża. Mogą być one uznawane za dane kategoryczne. Najpierw tworzymy słownik ze skończoną liczbą słów i traktujemy je, jako etykiety. Następnie w każdym zdaniu tworzymy wektory, oznaczają one, które wyrazy ze słownika są w zdaniu. Często takie podejście kończy się pojawieniem nowego słowa niezdefiniowanego wcześniej w słowniku. Metoda TFiDF [12, Chapter 3] to algorytmy o bardziej zaawansowanej naturze, które często wykorzystywana jest dla niewielkich zbiorów danych, innym sposobem jest wykorzystanie sieci neuronowych, których zadaniem jest przetworzenie danych tekstowych na wektory (np. modele word2vec). Metody wykorzystane podczas badań to TFiDF oraz architektura sieci neuronowej oparta na mechanizmie atencji - BERT. Forma, w jakiej są używane

oraz sposób zwracania przez nich wyników różni się od siebie. Jak można zaobserwować na rysunkach 2.1 oraz 2.2 głównych elementem różniącym obie metody jest to, że model BERT zwraca wektoryzację (zwane również 'osadzaniem' zdań w przestrzeni wektorowej) dla poszczególnych słów w ciągu, które najczęściej uśrednia się co prowadzi do zmniejszenia wymiarowości danych. Z macierzy $d \times n$ powstaje macierz $d \times 1$, gdzie d to wielkość wektora reprezentacji tekstu, a n to ilość słów w ciągu. Algorytm TFiDF natomiast zwraca reprezentację całego dokumentu, jako wektor o dużej liczbie wymiarów. Takie algorytmy przypisują każdemu nowemu słowu nowy wymiar przestrzeni. Dla wygodnego użycia tych danych wykorzystuje się algorytmy do redukcji wymiarowości wektorów.



Rysunek 2.1: Potok przetwarzania danych z wykorzystaniem algorytmu TFiDF



Rysunek 2.2: Potok przetwarzania danych z wykorzystaniem modelu BERT

2.2.1 Preprocesowanie tekstu

Zanim przekazany zostanie tekst do przetworzenia go na język zrozumiały przez model uczenia maszynowego, należy oczyścić go z niechcianych elementów. Częstym zjawiskiem jest, że próbki danych tekstowych zebrane do analizy zawierają znaki, które podczas wektoryzacji i analizy zwiększają złożoność problemu nie zmieniając istotnie wyniku. Zaczynając od rodzaju kodowania tekstu, przez wielkość liter, białe znaki, czy te niewystępujące w alfabecie łacińskim. Każdy zbiór jest inny i procesowanie powinno być indywidualnie rozpatrywane na podstawie analizy jego danych. Znaki interpunkcyjne, czy wartości liczbowe często pozostawia się w próbkach, lecz ich użycie podczas wektoryzacji danych staje się niezmiernie trudne.

Rysunek 2.3: Proces przetwarzania ciągu

Rysunek 2.4: Proces przetwarzania słowa

Listing 2.1 prezentuje metody użyte w procesie czyszczenia danych tekstowych. Proces ten zawarty jest także na rysunkach 2.3 dla przetwarzania ciągu oraz następujący po nim proces przetwarzania słowa - rysunek 2.4. Pierwszym etapem jest ujednolicenie wielkości liter, przez przetworzenie wszystkich wielkich na odpowiadające im małe litery. W danych często pojawiają się znaki, które nie są zawarte w kodowaniu ASCII. Kolejna metoda szuka sposobu na zamianę takich elementów na odpowiadające im znaki ASCII, a w przeciwnym wypadku usuwa je z tekstu. Po takim przefiltrowaniu dopuszczalnych znaków, następnym etapem jest usunięcie znaków interpunkcyjnych, które wprowadzają nieistotną złożoność do danych tekstowych. W przypadku używanego zbioru danych, jako znaki przydatne uznano '-' oraz '.', które nie są usuwane w tym etapie czyszczenia tekstu. Chociaż większość znaków interpunkcyjnych zostanie usunięta, pozostałe należy ujednolicić je do tych z tablicy ASCII - powtarzany jest krok, który wyszukiwał możliwości zmiany kodowania liter i znaków na te występujące w ASCII. Dodatkowo zmiana kodowania potrzebna jest także dla białych znaków, ponieważ nawet kodowanie spacji może mieć więcej niż dziesięć możliwości. Po takim oczyszczeniu tekstu, interpunkcji i białych znaków możliwe jest podzielenie dokumentu na tokeny - poszczególne słowa i praca na nich. Każde słowo z osobna przechodzi potok jego przetworzenia, w którym może zostać całkowicie odrzucone, lub wyłącznie oczyszczone do ostatecznej postaci. Na początku odrzucone zostaną znaki, które mogły pojawić się w tokenie, a nie powinny. Stworzona została ich lista zawierająca takie rzeczy, jako tabulatory, nawiasy i znaki cytowania. Następna metoda usuwa powtórzenia znaków w słowie, ponieważ często zdarza się, że w słowie, gdzie są dwie te same litery pod rząd, w formularzach znajdowały się trzy lub więcej. Przykładowo 'finally' stawało się 'finallly'. Trzy kolejne funkcje odrzucają słowa uznane za niepotrzebne w procesie tworzenia wektorów. Pierwsza z nich używa podstawowej list z biblioteki *NLTK* dla języka angielskiego , tak zwane 'Stop words'. Kolejna zajmuje się słowami o długości mniejszej niż 3, a ostatnio usuwa tokeny, w których występują wyłącznie znaki numeryczne. Jako ostatni krok procesowania wykonywana jest lematyzacja słów, co oznacza sprowadzenie wyrazów stanowiących odmianę danego zwrotu do podstawowej postaci, po czym wszystkie tokeny łączone są ponownie w ciąg tekstowy.

```
def text_process(input_string: str) -> str:
    txt = input_string.lower()
    txt = nlp.replace_non_latin_chars(txt)
    txt = nlp.spacing_punctuation(txt)
    txt = nlp.clean_special_punctuations(txt)
    txt = nlp.remove_spaces(txt)
    words = txt.split()
    words = nlp.remove_special_characters(words)
    words = nlp.reduce_lengthening(words)
    words = nlp.remove_stopwords(words)
    words = nlp.remove_short_words(words)
```

```
    words = nlp.remove_words_with_nums(words)
    words = nlp.find_words_lemmas(words)
    txt = ' '.join(words)
    return txt
```

LISTING 2.1: Kod python do procesowania pola formularza zgłoszenia

2.2.2 TFiDF

Preprocesowane dokumenty pozwalają na analizę całego zestawu danych. Oceną znaczenia słów w dokumencie i zbiorze zajmuje się algorytm TFiDF[12, Rozdział 3] - równanie 2.1. Tworzony jest słownik, czyli zbiór wszystkich słów występujących w zbiorze danych, na którym TFiDF bazuje. Jego nazwa rozwija się do - *term frequency times inverse document frequency*. Częstotliwość zwrotu (ang. term frequency) to liczba wystąpień poszczególnych tokenów (słowa zawartego w słowniku, któremu przypisane jest miejsce w wektorze) w dokumencie. Odwrócona częstotliwość w dokumentach (ang. inverse document frequency) oznacza odwróconą częstotliwość występowania tokenu w dokumencie na przestrzeni zbioru. Razem tworzą model statystyczny oceniający istotność słów na podstawie częstotliwości występowania. Częstotliwość występowania słowa w dokumencie z korpusu jest ważona przez odwrotną częstotliwość występowania w dokumentach korpusu. To wyłuskuje słowa rzadkie, ale istotne, które zazwyczaj niosą za sobą najwięcej informacji. Takim słowom przypisywana jest wysoka wartość. Równanie (2.1) przedstawia wzór obliczania współczynnika TFiDF dla danego słowa - s, dokumentu - d oraz zbioru dokumentów - D:

$$TFiDF(s,d,D) = TF(s,d)iDF(s,D) \tag{2.1}$$

Częstotliwość wystąpień (równanie (2.2)) to dana statystyczna operująca na pojedynczym słowie oraz pojedynczym dokumencie i określająca współczynnik wystąpień w porównaniu do ilości wyrazów w dokumencie. Równanie (2.2) przedstawia jego wzór dla danego słowa - s, dokumentu - d:

$$TF(s,d) = (f(s,d))/(\sum_{s' \in d} f(s',d)) \tag{2.2}$$

Odwrócona częstotliwość zwrotu - (równanie 2.4) w dokumentach to dana statystyczna operująca na całym zbiorze dokumentów, określająca jaka część wszystkich dokumentów zawiera wystąpienia danego tokenu. Jak dużo informacji jest dane słowo w stanie zapewnić w relacji do odróżniania dokumentów. Jeśli występuje on często jego znaczenie powinno spadać, dlatego użyto wariantu odwróconego tej metryki.

$$t_s = \sum_{d \in D\prime} 1 \tag{2.3}$$

Równanie 2.3 prezentuje liczbę dokumentów w których występuje dany token s, gdzie $(D\prime)$ to podzbiór D wyłącznie z dokumentami w których występuje s. W równaniu 2.4 D to zbiór dokumentów, a $|D|$ wielkość tego zbioru.

$$iDF(s,D) = \frac{|D|}{DF(s,D)} = log\frac{1+|D|}{1+t_s} \tag{2.4}$$

W praktyce statystyka ta podawana jest w skali logarytmicznej, a w mianowniku pojawia się dodawanie jedynki dla przypadku, gdy token nie znajduje się w żadnych z dokumentów, co prowadziłoby do dzielenia przez 0. Jako wynik każdemu dokumentowi przypisywany jest wektor. Najczęściej rzadki o długości równej długości słownika utworzonego na podstawie wszystkich dokumentów i ich tokenów znajdujących się w zbiorze. Miejsca w wektorze odpowiadają poszczególnym tokenom ze słownika dla słów występujących w danym dokumencie przypisywany jest ich współczynnik TFiDF, pozostałe elementy wypełniane są zerami. Tak stworzona reprezentacja prezentuje dokument tekstowy, jako dane kategoryczne dodatkowo zawierając metrykę znaczenia dla każdego tokenu, który wystąpił.

2.2.3 Koder - dekoder

Architektura koder-dekoder wykorzystywana jest w maszynowym tłumaczeniu tekstów, rozpoznawaniu mowy, czy tytułowaniu filmów. W przypadku tłumaczenia tekstów, jej danymi zarówno wejściowymi i wyjściowymi jest dokument tekstowy, na przykład w innym języku. Jak przedstawia rysunek 2.5 koder używany jest do wektoryzacji dokumentu, a dekoder do ponownej zamiany na tekst. Jako, że problem prezentowany w tej pracy odnosi się do rozpoznawania duplikatów, przy zastosowaniu wektorów dokumentów tekstowych wyłącznie część kodera jest wykorzystywana w tym procesie jak i modelu BERT'a, który jest wykorzystywaną implementacją tej architektury.
Transformer składa się z zestawionych ze sobą identycznych części łączących równolegle połączone warstwy mechanizmu atencji (eng. self-attention) w wariancie multi-head oraz następujące po nich w pełni połączone warstwy neuronów. Wynik każdej z tych warstw jest agregowany z danymi wejściowymi oraz przekazywany do warstwy normalizacyjnej. Każda z podwarstw zwraca ten sam rozmiar wektora, ponieważ do agregacji wykorzystuje się sumowanie względem obu wektorów według indeksów. Przed wykorzystaniem warstw atencji, model zmienia dane wejściowe na wektory (embeddings), a także wykonuje kodowanie pozycji. Ma ono oznaczać wektory słów z zależności od ich pozycji w dokumencie. Edytuje on wartości wektora słowa tak by model potrafił wykorzystać informację o tym, na którym miejscu w dokumencie token wystąpił. Algorytmy kodujące pozycje mogą wykorzystywać proste funkcje matematyczne, jak i warstwy neuronowe. Funkcje przedstawione w równaniach (2.5) oraz (2.6) użyte zostały w opisie implementacji [18]. Określają one jaka wartość powinna zostać dodana to każdego z parametru wektora słowa, 'pos' określa pozycję słowa w ciągu, a 'i' - element w wektorze danego słowa, d_{model} to liczba wymiarów słowa tworzona przy osadzaniu w przestrzeni wektorowej.

$$PE_{(pos,2_i)} = sin(pos/10000^{2i/d_{model}}) \tag{2.5}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{2.6}$$

2.2.4 Warstwy atencji

Mechanizm atencji [18] to główny składnik architektury Transformera. Wariancja multi-head w nim używana to połączenie równoległe n takich samych warstw atencji. Warstwa atencji do stworzenia wyniku

RYSUNEK 2.5: Architektura koder-dekoder

używa 3 macierzy:

Q - nazywana macierzą zapytań, jest reprezentacją danych wejściowych

K - nazywana macierzą kluczy, to reprezentacja wektorów, wobec których obliczamy atencję, sprawdzone zostaje, jakie znaczenie mają elementy zawarte w tej macierzy względem elementów macierzy Q

V - macierz wartości, na które obliczona atencja jest nakładana, ich wektor jest zwracany przez warstwę po modyfikacji przemnożeniu go przez macierz atencji

W zależności od zastosowania K oraz V może być zainicjalizowana tak samo jak Q, co pozwala na analizę atencji wewnątrz wektora słów (self-attention), pomiędzy tokenami, które się w niej znajdują. Atencję można obliczać w odniesieniu do zestawu innych tokenów, na przykład słów kluczy występujących w zbiorze. Dla każdej z trzech macierzy wejściowych tworzone są macierze wag, które przechowują informacje zyskane w procesie propagacji wstecznej. Macierze oznaczone literami Q,K,V oznaczają dane wejściowe przemnożone przez odpowiednie macierze wag co dokładniej widać we wzorze (2.9).

$$Attention(Q, K, V) = Softmax((Q * K^T)/\sqrt{d_k}) * V \tag{2.7}$$

Równanie (2.7) przedstawia proces obliczania atencji, jako pierwszy krok wykonywane jest mnożenie macierz Q z transponowaną macierzą K, dzielenie przez d_k (d_k to wymiar macierzy kluczy) zostało wprowadzone w kontrze do występowania małego gradientu. Po przekazaniu rezultatu do funkcji softmax wykonywana jest operacja, jako suma wag, gdzie każda wartość macierzy wag mnożona jest z odpowiadającą wartością w macierzy wyjścia softmax. Operacja ta zwraca współczynnik atencji pomiędzy dwoma tokenami. Atencja wewnętrzna (self-attention) w transformerze zastępuje wcześniej używane warstwy rekurencyjne, czy konwolucyjne stając się nową architekturą wiodącą prym w kontekście kodowania tekstu. Każda z tych warstw wyróżnia się nad w pełni połączoną warstwą neuronową przez wykorzystanie informacji o położeniu tokenów względem siebie. Pozwala to sieciom na semantyczne zrozumienie zdania oraz jego kontekstu, co nie jest możliwe przy analizie każdego słowa osobno, bez zaznaczenia jakichkolwiek zależności między nimi. Sieci rekurencyjne przechowują informację o poprzednich elementach, konwolucyjne o najbliższym sąsiedztwie, natomiast warstwy atencji robią krok dalej analizując ciąg słów w całości.

Mechanizm atencji w wariancie multi-head przedstawiona we wzorach (2.8) oraz (2.9) to architektura wykorzystująca warstwy atencji połączone równolegle. Wewnątrz każdej z podwarstw atencji obliczany jest wynik funkcji softmax z przemnożonych macierzy zapytań i transponowanej macierzy kluczy, znormalizowanych przez pierwiastek z długości wektora kluczy. Zanim jednak wynik tej operacji zostanie pomnożony

przez macierz wartości, wektory wynikowe ze wszystkich podwarstw są łączone w jeden.

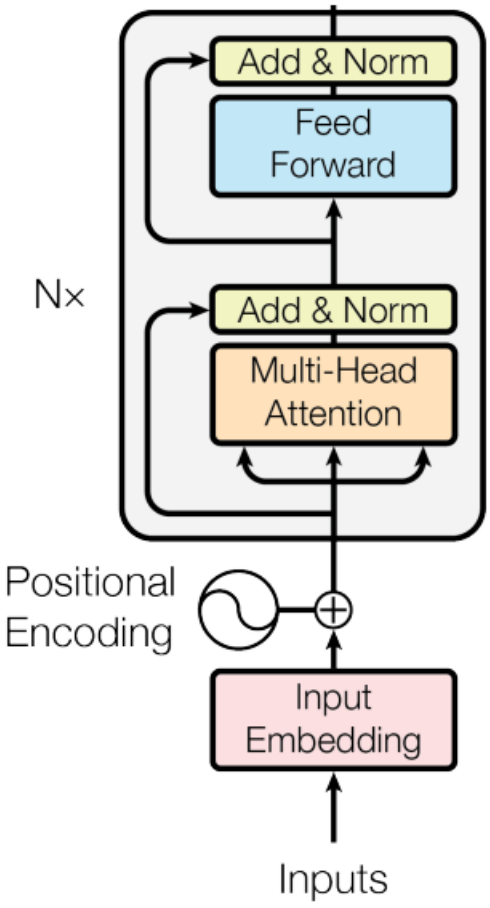
$$MultiHead(Q, K, V) = Concat(head1, ..., headh)W^O \tag{2.8}$$

gdzie

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.9}$$

2.2.5 BERT

BERT [9] to model wektoryzujący tekst stworzony w *Google*, którego nazwę rozwija się do "Bidirectional Encoder Representations from Transformers". Bazując na części kodera architektury Transformera [18] zaprezentowanego na rysunku 2.6.



Rysunek 2.6: Architektura kodera [18]

Modele głębokie wektoryzujące dane mają swoistą blokadę wielkości przy pracy na niewielkich zbiorach danych. Duże modele wymagają większej ilości przykładów i dłuższego czasu treningu tak, by każdy z parametrów znalazł swoje odwzorowanie i kontrybuował do predykcji właściwego wyniku. Znaleziono jednak sposób na obejście tej przeszkody. Słowo pisane ma tą zaletę, że niezależnie od tego, na czym uczymy się czytać, znaczenie słów pozostaje praktycznie niezmienne, a nowe zdania potrafimy przyswoić z małym trudem. Nawet, jeśli naszym celem jest rozpoznawanie rodzaju kwiatów po opisie, to ogólna znajomość

dokumentów w danym języku pozwala na uzyskanie przewagi jeszcze przed przystąpieniem do zaznajamiania się z tym konkretnym zadaniem. Taki sam pomysł użyto podczas trenowania ogromnych modeli do wektoryzacji tekstu, korpus dokumentów nie musi a nawet nie powinien mieć konkretnego przeznaczenia, czy opisywać danej dziedziny. Model trenowany jest na jak największym zbiorze reprezentującym relatywnie wszystkie możliwe użycia słów i zwrotów, by potem zostać dostosowany do konkretnego zadania - nazywane dostrajaniem. Co wyróżnia model Transformera to użycie warstw atencji, które pozwalają na ocenę zależności między słowami w dokumencie, czyli analizę semantyczną.
Architektura modelu składa się z wielu warstw dwukierunkowego kodera z architekturą Transformera opisanej w [18]. Liczba warstw jak i innych parametrów zależy od wersji modelu, który został udostępniony. Do użytku przekazano modele trenowane w różnych językach, czy na korpusach wielkojęzykowych. Model podstawowy zawiera 12 warstw transformera, czyli szeregowo połączone ze sobą architektury kodera zawierające 12 warstwami self-attention (opisane poniżej). Zwracają one reprezentację wektorową o długości 768. BERT zapewnia także warstwę preprocesowania tekstu, przekazywany dokument, czy też zdanie przetwarzane zostaje przez embeddingsy WordPiece ze słownikiem 30 000 tokenów. Token '[CLS]' dodawany jest na początek każdego dokumentu, dodatkowa logika używania jest dla przekazywanych par pytanie-odpowiedź które także są przyjmowane przez model [9, chapter 3]. Dla każdego dokumentu jest tworzona reprezentacja wektorowa i przekazywana do kodera.
Użycie wytrenowanego modelu wymaga wspomnianego już wcześniej dostrajania. W najprostszej postaci polega on na umożliwieniu trenowania odpowiedniej liczbie warstw transformera (lub wszystkich) opisanych w punkcie 2.2.3 i z użyciem własnego zbioru danych dopasowanie parametrów do niego. Operację taką wykonuje się na niskim współczynniku uczenia (najczęstszym jest 10^{-5}), co jest w szczególności istotne dla małych zbiorów danych. Otworzenie warstw pozwala na dopasowywanie ich parametrów w procesie uczenia, użycie własnego zbioru danych daje możliwość dostrojenia otwartych warstw do tego zestawu danych. Mały współczynnik uczenia wybierany jest z powodu użycia mniejszego zbioru danych niż przewidziany jest dla modelu z tak dużą liczbą parametrów. Z tego również powodu, często część warstw pozostawia się zamknięte(ang. bagging), ponieważ nawet przy tak małym współczynniku uczenia model przesadnie dopasowuje parametry do przypadków w zbiorze. Zjawisko to nazywane nazywane jest przetrenowaniem modelu.

2.3 PCA

PCA (ang. principal component analysis) to algorytm statystyczny, który pozwala na zmniejszenie wymiarowości wysokowymiarowych danych, odwzorowując dane na przestrzeń o mniejszej liczbie wymiarów, przy zachowaniu jak największej liczby informacji w zbiorze. Zmniejszenie ilości wymiarów reprezentujących dane ma swoją cenę, algorytmy jak PCA dostarcza statystykę ile informacji stracimy, w przypadku wyboru konkretnego wymiaru przestrzeni. Zbiory danych o mniejszej wymiarowości są łatwiejsze do analizy (przez niższą wymiarowość czas ich przetworzenia przetwarzania przez algorytm zmniejsza się) i wizualizacji. PCA wyszukuje wariancje na osiach ortogonalnych, czyli działa liniowo. Algorytm diagonalizuje macierz wariancji danych tworząc transformacje do nowej przestrzeni, której osiami są wektory

własne macierzy wariancji. Wektory własne sortowane są względem wartości własnej im odpowiadających. Przed przystąpieniem do analizy poszczególnych wymiarów należy ustandaryzować rozkład zbioru danych tak by każda zmienna w podobny sposób wpływała na wyniki analizy. Ponieważ PCA w swoim założeniu działa na wariancji(czyli zmienności sygnału) pomiędzy elementami, standaryzacja przed jej obliczeniem wpływa na odpowiednie traktowanie zmiennych - stają się one znormalizowane, by sposób ich interpretacji był ujednolicony. Zakres cechy pomiędzy 0, a 100 oraz te pomiędzy 0, a 1 będą tak samo ważne. Jak przedstawia równanie (2.10), do standaryzacji wartości x należącego do cechy 'i', użyta zostaje średnia - a_i oraz odchylenie standardowe cechy - σ_i. Dane są standaryzowane do średniej - 0 i wariancji równej 1.

$$x'_i = \frac{(x_i - a)}{\sigma} \tag{2.10}$$

Po ustandaryzowaniu danych przychodzi moment na ocenę zmienności danych wejściowych w rozumieniu obliczenia ich kowariancji na przestrzeni zbioru. Macierz kowariancji - na równaniu (2.11), przykład dla trzywymiarowej przestrzeni - pozwoli znaleźć zależności pomiędzy wymiarami, oceniając czy są one skorelowane ze sobą co oznaczałoby że przechowują podobne informacje. Macierz kowariancji jest kwadratową macierzą symetryczna o wymiarach równych ilości zmiennych w próbce danych (wymiarowi wektora).

$$Macierzkowariancji = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix} \tag{2.11}$$

Równanie (2.12) przedstawia wzór obliczenia kowariancji dla zmiennych X i Y, wykorzystując wartość oczekiwaną - E.

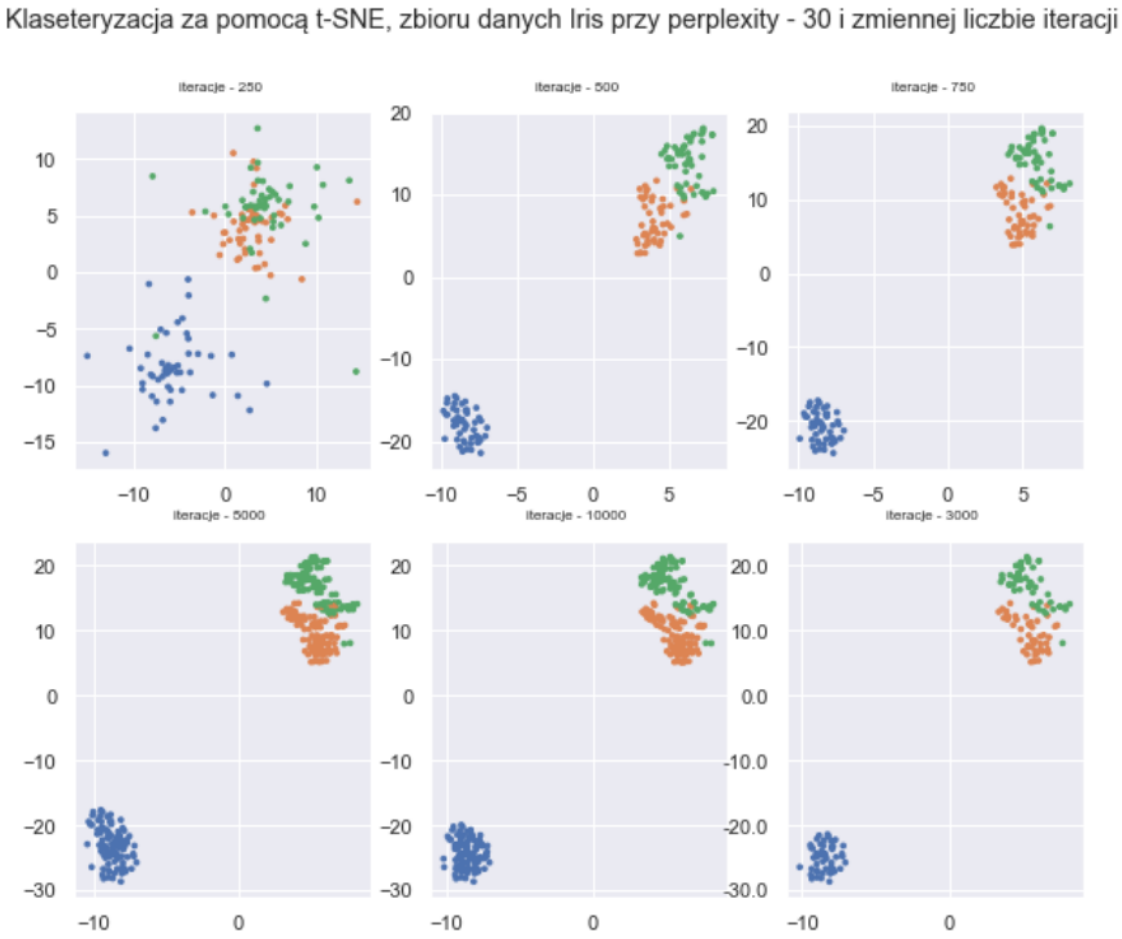
$$cov(X,Y) = E(X * Y) - [E(X) * E(Y)] \tag{2.12}$$

Jeśli wartość współczynnika kowariancji jest dodatnia dla dwóch zmiennych (np. wymiarów n,m w próbkach) oznacza to, że są one ze sobą skorelowane (jeśli wartość wymiaru n rośnie to m także rośnie). W przypadku, gdy wartość współczynnika kowariancji jest ujemna - zmienne są odwrotnie skorelowane. Im większa wartość tym mocniejsza korelacja. Korzystając z macierzy relacji (określającą korelacje między zmiennymi) możemy użyć jej do określenia komponentów (wymiarów) o najważniejszym statusie. Tych, które mają największy wpływ na rozróżnienie próbek. Operacja ta wykonywana jest poprzez obliczenie wektorów własnych i wartości własnych dla macierzy kowariancji (które odpowiadają konkretnych elementom wektora próbek zbioru danych). Jest to idea wywodząca się z algebry liniowej, która pozwala określić składowe w wektorach danych, które najlepiej opisują zbiór. Składowe główne (najważniejsze) powstają, jako wektory własne. Nowo utworzone zmienne w sposób nieskorelowany ze sobą opisują informacje zawarte w pierwotnej próbce. PCA produkuje taką samą liczbę głównych składowych, co wymiar wektora wejściowego (tyle powstaje wektorów własnych z macierzy), lecz te najbardziej znaczące są na pierwszych miejscach w wektorze (posortowane są w kategorii ważności dla opisu zbioru danych - według wartości własnych przynależnych do wektorów własnych). Zmniejszenie wymiarowości danych polega na wybraniu k pierwszych wymiarów wektora wynikowego, gdzie $k < d$ (d - wymiar wektorów wejściowych), przy zachowaniu jak największej wariancji (polecane jest 70% lub więcej).

2.4 Truncated-SVD

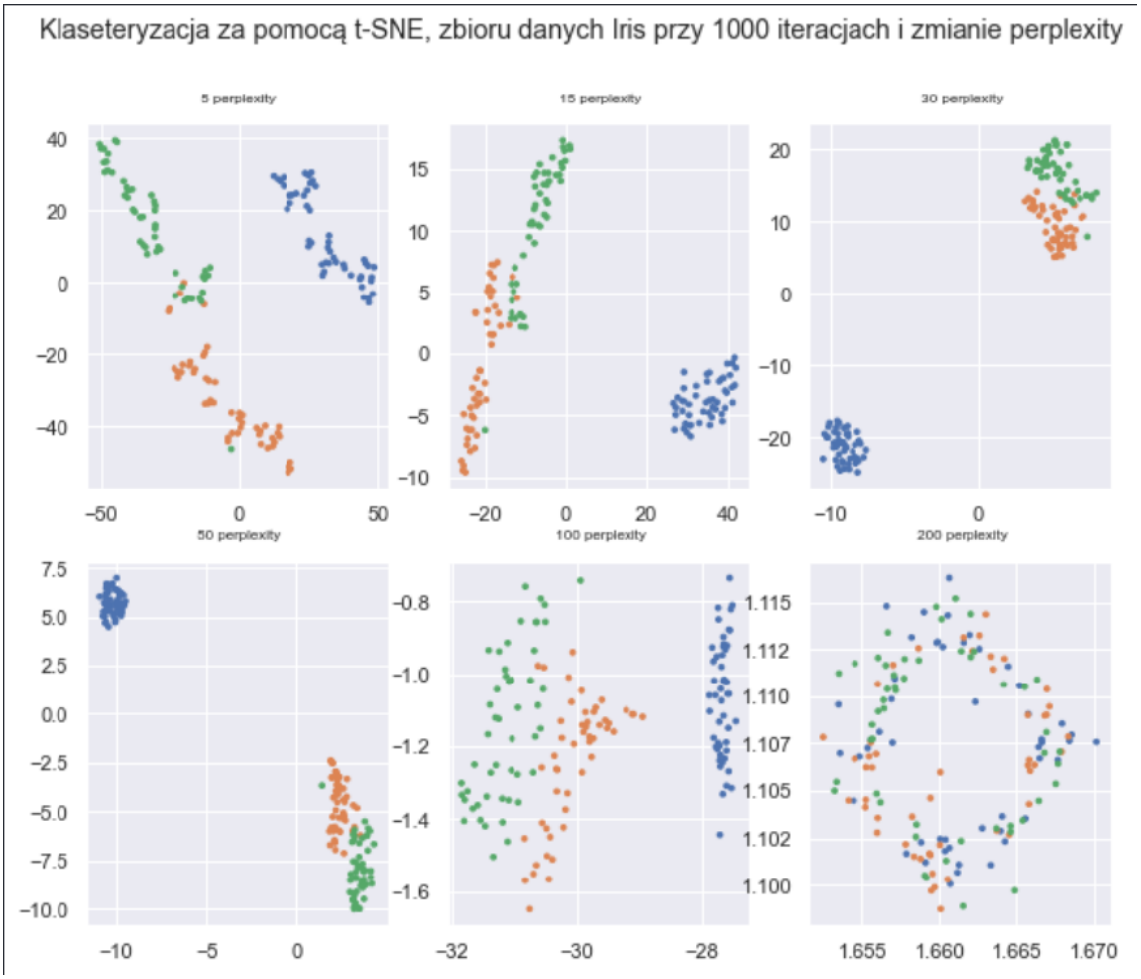
Trunkated-SVD (obcięty rozkład na wartości osobliwe) [16] [12], jak i PCA należy do zbioru metod analizujących ukryte znaczenie semantyczneo - LSA (ang. Latent semantic analysis). Podobnie jak PCA proponuje ono algorytm redukowania wymiarowości danych w sposób liniowy. Dzieje się to za pomocą analizy średnich dla skróconych dekompozycji głównych składowych (SVD), jest on rekomendowany do pracy z wektorami TFiDF. Tak samo jak poprzednia metoda, ta zwraca k-elementowy wektor wyjściowy dla d-elementowego wektora wejściowego, gdzie $k < d$, czyli zwraca wektor reprezentujący te same dane o mniejszym wymiarze. Analizy wyników używania tej metody pokazały, że radzi sobie ona bardzo dobrze z problemem występowania synonimów i polisemów. Synonim jest to wyraz równoważny znaczeniowo innemu wyrazowi na tyle by można go zastąpić w danym kontekście, natomiast polisemy to wyrazy o wielu znaczeniach powiązanych ze sobą. Równania opisujące tę metodę wyjaśnione są w [16].

2.5 t-SNE



Rysunek 2.7: Przykład zastosowania t-SNE ze zmiennym współczynnikiem liczby kroków algorytmu

t-SNE (t-Distributed Stochastic Neighbor Embedding) to technika zmniejszania wymiarowości zaproponowana przez L. v.d. Maaten oraz G. Hinton w 2008 roku [17]. Zazwyczaj wykorzystuje się ją do redukcji



Rysunek 2.8: Przykład zastosowania t-SNE ze zmiennym współczynnikiem *nieokreśloności*

wymiarowości danych do 2 lub 3 wymiarów, by umożliwić wizualizację na płaszczyźnie, czy w przestrzeni i analizę powstałych klastrów. Redukcja odbywa się przy zachowaniu zależności między nimi. Metoda jest nieliniowa i adaptuje się do bazowych danych wykonując inne transformacje w różnych regionach, co może powodować problemy w interpretacji wyników. Głównym współczynnikiem, na którym bazuje algorytm jest *nieokreśloność* [3] (ang.*perplexity*) - dostosowywane przez użytkownika. Ma ono znaczenie w kontekście skupienia się na zachowaniu zależności danych na poziomie lokalnym i globalnym zbioru. W praktyce jest to liczba bliskich sąsiadów, którą przewidujemy, że punkt będzie miał. W oryginalnej pracy [17] autor podkreśla znaczenie tego współczynnika oceniając, że typowe dla niego wartości oscylują pomiędzy 5 a 50 i na pewno powinno być mniejsze niż liczba punktów w zbiorze danych. Jedynym sposobem analizy czy zadana *'nieokreśloność'* jest prawidłowa, jest porównywanie wykresów tych samych próbek danych dla różnie zadanego parametru. Kolejnym parametrem, który algorytm przewiduje jest liczba kroków - iteracji algorytmu. Każdy kolejny krok powoduje kolejną transformację zbioru danych wpływając na końcowy wygląd wektorów próbek. Najczęściej kroki wykonuje się do zaobserwowania stabilizacji w kolejnych iteracjach. Wartość zależna jest od używanego zbioru danych i nie ma przewidzianego zakresu. Algorytm t-SNE bazuje na zaprezentowanym w 2002 roku przez Hintona i Roweisa SNE, szerzej przedstawione w [10]. Przykład porównania wyników przy zmiennej *nieokreśloności* można zaobserwować na

rysunku 2.8, gdzie każdy z wykresów prezentuje zbiór danych Iris o zmniejszonej wymiarowości wektorów reprezentacji do 2 (osie x i y przedstawiają wartości dla tych wymiarów). Algorytm t-SNE przetwarza zbiór każdorazowo ze stałą liczbą iteracji i zmieniającym się współczynnikiem *nieokreśloności* od 5 do 200. Ten sam zbiór danych wykorzystano do porównania wyników algorytmu t-SNE przy zmiennej liczbie iteracji na rysunku 2.7. Te wykresy także prezentują zbiór danych o zmniejszonej wymiarowości wektorów reprezentacji do 2. Algorytm przetwarza zbiór każdorazowo ze stałą liczbą *nieokreśloności* i zmieniającym się współczynnikiem iteracji od 250-3000.

Pierwszym krokiem SNE jest zmiana wielowymiarowych odległości Euklidesowych pomiędzy punktami na prawdopodobieństwa warunkowe, pokazujące jak próbki są do siebie podobne. Dla x_i oraz x_j, podobieństwo jest obliczane jako prawdopodobieństwo warunkowe $p_{j|i}$, które jest zdefiniowane jako prawdopodobieństwo że x_i uznałoby x_j za swojego sąsiada. Dla podobnych do siebie próbek przypisywana jest wysoka wartość prawdopodobieństwa, niższa wartość dla mniej podobnych próbek. W podobny sposób prawdopodobieństwa przypisywane są dla próbek w mniejszej przestrzeni wymiarowej.

SNE stawia sobie za cele by te dwa rozkłady dopasować do siebie z jak największą dokładnością. Służy do tego utylizacja funkcji straty (2.13). Funkcja skupia się na odzyskaniu lokalnej struktury pomiędzy elementami (dystansów) po rzutowaniu na przestrzeń o mniejszej liczbie wymiarów, czyli zachowaniu jak najlepiej relacji pomiędzy próbkami.

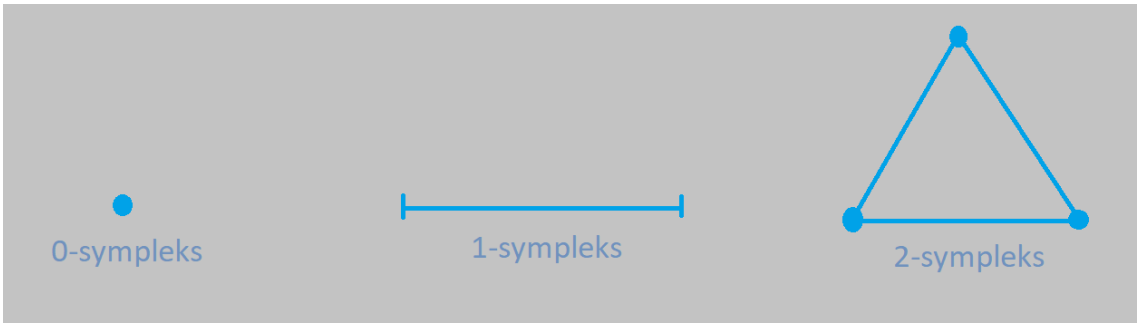
$$C = \sum_i KL(P_i||Q_i) = \sum \sum p_{j|i} * \log p_{j|i}/q_{j|i} \tag{2.13}$$

$$d_K L(p,q) = \sum_i p(i) * \log_2 p(i)/q(i) \tag{2.14}$$

Strata definiowana jest jako suma dywergencji Kullbacka-Leiblera pomiędzy wszystkimi punktami zbioru danych. Dywergencja Kullbacka-Leiblera wykorzystywana jest w statystyce właśnie do określania rozbieżności w rozkładach prawdopodobieństw i dla rozkładów p oraz q definiowana jest wzorem (2.14). Minimalizacja odbywa się przy użyciu metody gradientu prostego (ang. gradient descent). Przez to, że rozbieżność KL nie jest symetryczna, różne rodzaje błędów pojawiają się dla poszczególnych dystansów pomiędzy parami.

2.6 UMAP

UMAP (Uniform Manifold Approximation and Projection) [14], [13] to kolejny algorytm do redukcji wymiarowości danych z kategorii technik *manifold learning*. UMAP tworzy wielowymiarowy graf reprezentujący dane, by potem zoptymalizować go do przestrzeni o mniejszej ilości wymiarów, z zachowaniem struktury i zależności między danymi. Tworzenie grafu polega na użyciu 'rozmytego kompleksu symplicjalnego'. Jest to przedstawienie grafu z wagami, które reprezentują prawdopodobieństwo połączenia dwóch punktów. Do stworzenia grafu połączeń używa się prostych bloków o nazwie sympleksy. Są to najprostsze wypukłe figury geometryczne. W UMAP są one tworzone przy użyciu k+1 punktów w danym otoczeniu, łącząc je ze sobą. K-wymiarowy sympleks nazywany jest k-sympleksem. 0-sympleks to punkt, 1-sympleks to odcinek, a 2-sympleks to trójkąt.



Rysunek 2.9: Przykład wizualizacji sympleksów na podstawie [14]

Mając podstawowe narzędzie do budowy grafu, musimy być w stanie łączyć je razem tak by powstały odpowiednie zestawienia. Do tego służy kompleks symplicjalny (eng. simplicial complex). Jest to zestaw sympleksów tak połączonych, aby ściany do siebie pasowały. Aby zastosować te narzędzia do zbioru danych algorytm używa kompleksu Cech'a, szerzej wyjaśniony w [1]. Warunkiem połączenia punktów jest to, że znajdują się w swoim sąsiedztwie. Najprostszym sposobem jest ustawienie maksymalnego promienia odległości, w którym punkty uznawane są za sąsiadów, lecz UMAP wykorzystuje bardziej złożone rozwiązanie. Wywodzi się to z faktu, że zbiory danych często mają nierównomiernie rozłożone punkty, gęstość zmienia się w różnych miejscach przestrzeni. Możliwym jest wtedy, że niektóre punkty nie miałyby żadnych sąsiadów, gdy inne znajdowałyby się w sąsiedztwie ogromnej liczby punktów. Dla niektórych zbiorów byłoby to niemożliwe, by zbalansować zadany promień tak by tworzony graf był optymalny. W przypadkach skrajnych wszystkie elementy byłyby połączone ze wszystkimi, lub tworzone byłyby niezależne od siebie klastry. Przy analizie zastosowania algorytmu na zbiorach danych twórcy doszli do wniosku, że punkty muszą mieć obliczany indywidualny współczynnik promienia w zależności od ich sąsiedztwa. Wynikiem tego jest obszar zawierający k najbliższych sąsiadów dla każdego z punktów, więc w praktyce zamieniamy ustalenie stałego promienia na ustalenie stałej liczby sąsiadów. Wybór małej wartości pozwala skupić się na lokalnej strukturze danych, gdy wysoka wartość daje możliwość skupienia się na ogólnym rozłożeniu z pominięciem delikatnych szczegółów. Skupienie się na ogólnym rozłożeniu sprawia, że algorytm staje się bardziej uniwersalny, co może być pomocne w przypadku potrzeby użycia go do redukcji wymiarowości nowych próbek danych bez powtórnego trenowania. Dla rozróżnienia znaczenia połączeń wprowadza się pojęcie potencjału połączenia. Prowadzi to do zamiany binarnej reprezentacji krawędzi (1-istnieje, 0-nie istnieje) na wartości rozmyte w przedziale 0-1. Swoiste prawdopodobieństwo istnienia połączenia będzie zmniejszało się zaczynając od pierwszego sąsiada wraz z odległością od punktu. Wiedząc, że promienie różnią się w zależności od punktów, wartości potencjału połączenia między punktami x_i i x_j oraz x_j i x_i także będą często różne. Proponowanym rozwiązaniem jest agregacja wartości:

$$w_{x_i x_j} = w_{x_i \to x_j} + w_{x_j \to x_i} - w_{x_i \to x_j} * w_{x_j \to x_i}, \tag{2.15}$$

gdzie $w_{x_i \to x_j}$ i $w_{x_j \to x_i}$ to wartości krawędzi skierowanych pomiędzy punktaki x_i i x_j. Wynikiem jest waga nieskierowanej krawędzi $w_{x_i x_j}$ pomiędzy tymi samymi punktami. Dla optymalizacji działania implementacje UMAP, często wykorzystują kompleks Vietoris-Rips w miejscu kompleksu Cech'a, który przywiduje

tworzenie wyłącznie 0- i 1-sympleksów.

Stworzony graf z krawędziami i ich wartościami optymalizowany jest w ten sam sposób, co punkty i dystanse w algorytmie t-SNE.

2.7 Las losowy

Las losowy (eng. random forest) to algorytm klasyfikacyjny polegający na połączeniach wielu drzew decyzyjnych. Każde z drzew oddzielnie zwraca swoją predykcję, która przetwarzana jest by zwrócić ostateczny wynik.

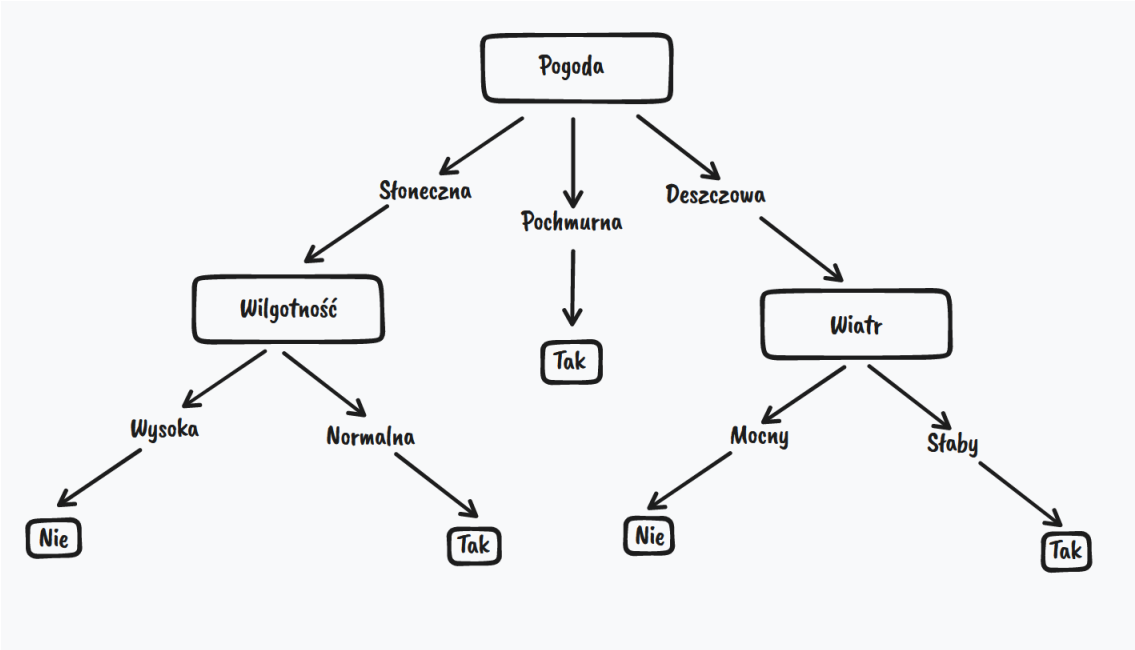
Drzewa decyzyjne to nieparametryzowana metoda uczenia maszynowego wykorzystywana w problemach regresji i klasyfikacji. Celem jest stworzenie modelu, który przewiduje wartość próbki na podstawie wyuczonych decyzji odnoszących się do wartości zawartych w wektorze opisującym dane. Model odnajduje elementy, których wartości są wyznacznikiem, jaki wynik powinien zostać zwrócony i tworzy kolejne stopnie drzewa (warunki w procesie przetwarzania danych wejściowych na wynik modelu), aż jest gotowy przewidywać wynik.

Dzień	Pogoda	Temperatura	Wilgotność	Wiatr	Odbędzie się trening?
1	Słonecznie	Gorąco	Wysoka	Słaby	Nie
2	Pochmurnie	Gorąco	Wysoka	Słaby	Tak
3	Słonecznie	Umiarkowana	Normalna	Mocny	Tak
4	Pochmirnie	Umiarkowanie	Wysoka	Mocny	Tak
5	Deszczowa	Umiarkowanie	Wysoka	Mocny	Nie
6	Deszczowa	Chłodno	Normalna	Mocny	Nie
7	Deszczowa	Umiarkowanie	Wysoka	Słaby	Tak
8	Słonecznie	Gorąco	Wysoka	Mocny	Nie
9	Pochmirnie	Gorąco	Normalna	Słaby	Tak
10	Deszczow	Umiarkowanie	Wysoka	Mocny	Nie

TABLICA 2.1: Tablica danych dla przykładowego drzewa decyzyjnego [16]

Na rysunku 2.10 widzimy przykład modelu oceniającego czy trening sportowy powinien się odbyć ze względu na warunki atmosferyczne, na podstawie danych zaprezentowanych w tablicy 2.1. Przekazujemy takie informacje jak: pogoda, temperatura, wilgotność i wiatr, które mogą zostać użyte przez model w zapytaniach warunkowych. Sprawdzenie poprawności modelu odbywa się przez porównanie wyników do kolumny 'Odbędzie się trening?'. Zaprezentowane drzewo o głębokości 2, wybiera trzy pola: pogoda, wilgotność i wiatr na podstawie, których może zwrócić najpewniejsze wyniki.

Drzewa decyzyjne charakteryzują się prostotą i prędkością obliczania, co pozwala na wykorzystanie zbioru takich modeli. Jedno drzewo samo w sobie nie jest w stanie zawrzeć wszystkich informacji z konkretnego zbioru danych by z dużą pewnością przewidywać jego wyniki (chyba że zbiór jest naprawdę prosty), równoległe ich tworzenie pozwoli wykorzystać wiele scenariuszy predykcji. Jednak stworzone drzewa muszą się od siebie różnić. Stosuje się do tego dwie metody, pierwsza to użycie tylko części zbioru danych. Losowo można zmieniać stosunki występowania danych etykiet, lub w ogóle wybrać tylko część dla danego drzewa (choć taki zabieg może mieć duże znaczenie na wyniki). Mniej inwazyjnym sposobem jest losowe dobranie próbek bez żadnych obostrzeń. Drugim zabiegiem pozwalającym na zmianę wyglądu modeli drzew losowych jest selektywny dobór cech danych przekazywanych na wejściu. Nie wszystkie



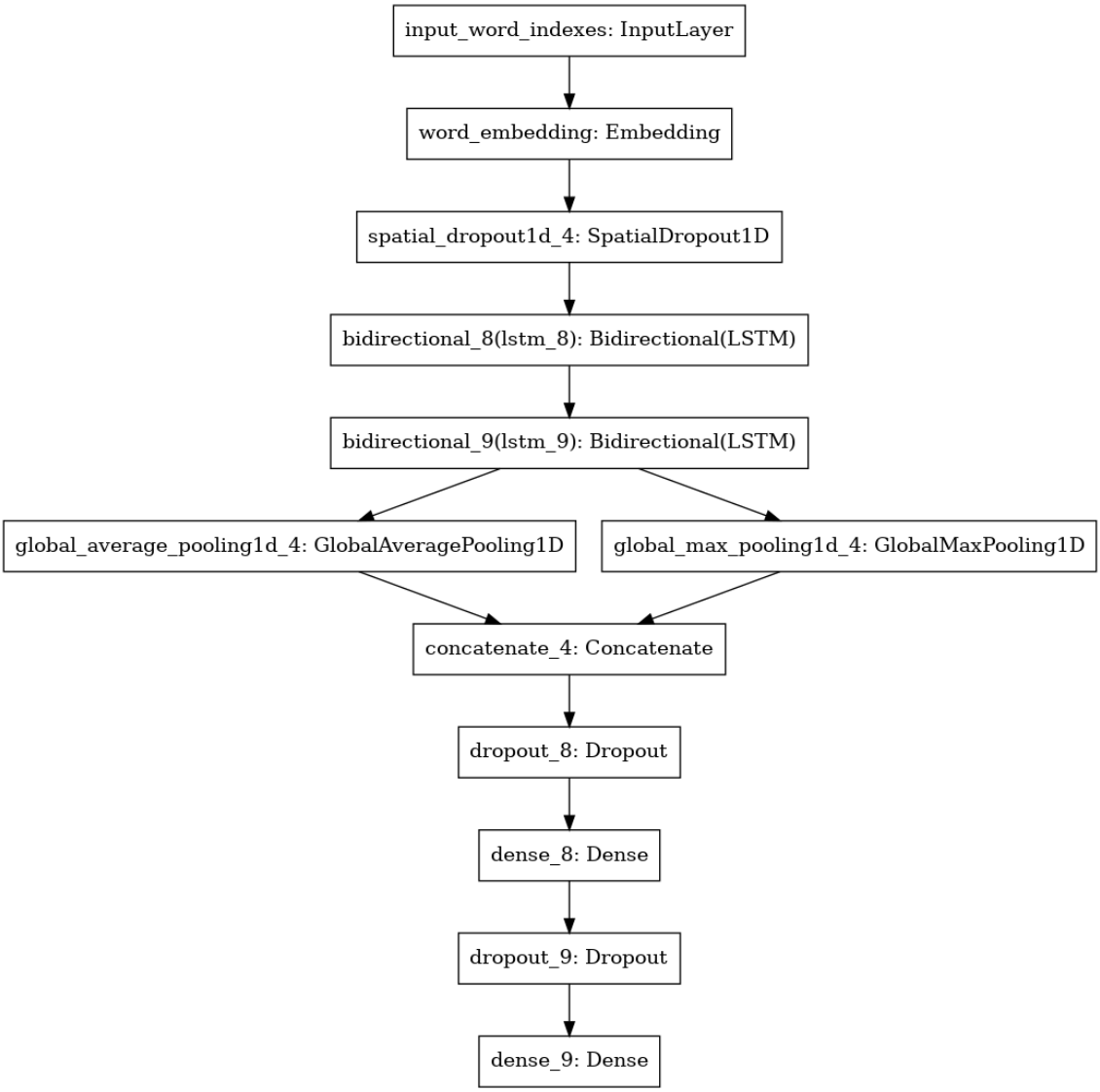
RYSUNEK 2.10: Przykład drzewa decyzyjnego na podstawie [16]

cechy używane do predykcji zmieniają kroki drzewa do zwrócenia wyniku. Tak stworzony las różnych drzew pozwala na spojrzenie na problem z różnych perspektyw. Las losowy przy wykorzystaniu, jako algorytm regresji zwraca wynik, jako średnia wyników drzew decyzyjnych. Jeśli chodzi o dobór cech dla różnych drzew to powstała konwencja dla problemów regresyjnych, gdzie każde drzewo przyjmuje 1/3 ze wszystkich dostępnych, a w problemach klasyfikacji za podstawową ilość uznaje się pierwiastek z dostępnych cech. Jednak parametr ten warto modyfikować, jeśli uznamy, że nie odpowiada on używanemu zbiorowi danych. Ważnym jest także wybór odpowiedniej głębokości drzewa, oczywiście dla danych o małej złożoności nie musi być on wysoki. Przy użyciu algorytmu lasu losowego ten parametr zazwyczaj pozostaje w granicach 1-10. Ostatnią decyzją do podjęcia jest liczba drzew decyzyjnych, które chcemy użyć, ważnym jest by ten parametr pasował do złożoności danego zbioru danych i ich cech oraz ilości próbek w zbiorze, ponieważ także one będą wybierane dla poszczególnych drzew.

2.8 Modele głębokie uczenia maszynowego

Głębokie sieci neuronowe to część bardziej obszernego obszaru - uczenia maszynowego [16] [15] [6] [8]. Przeznaczone dla większych zbiorów i złożonych danych, algorytmy bazującą na ideii działania przypominającej działanie ludzkiego mózgu. W podstawowej koncepcji takie modele składają się z następujących po sobie warstwach neuronów. Konfiguracja takich parametrów następuje w procesie uczenia, gdzie próbki danych przechodzą przez kolejne warstwy, aż ostatnia zwraca wynik modelu. Takowy wynik porównywany jest z oczekiwanym, by obliczyć stratę (jak bardzo model się pomylił). Proces propagacji wstecznej wykorzystuje obliczoną stratę, by aktualizować wagi (nowa wartość neuronów obliczana jest w zależności od wybranego algorytmu) do wartości, która ma minimalizować występującą stratę. Rysunek 2.11 przedstawia przykładowy wygląd modelu zwracającego wynik jako warstwa z neuronami i przyjmująca

jako dane wejściowe tekst.

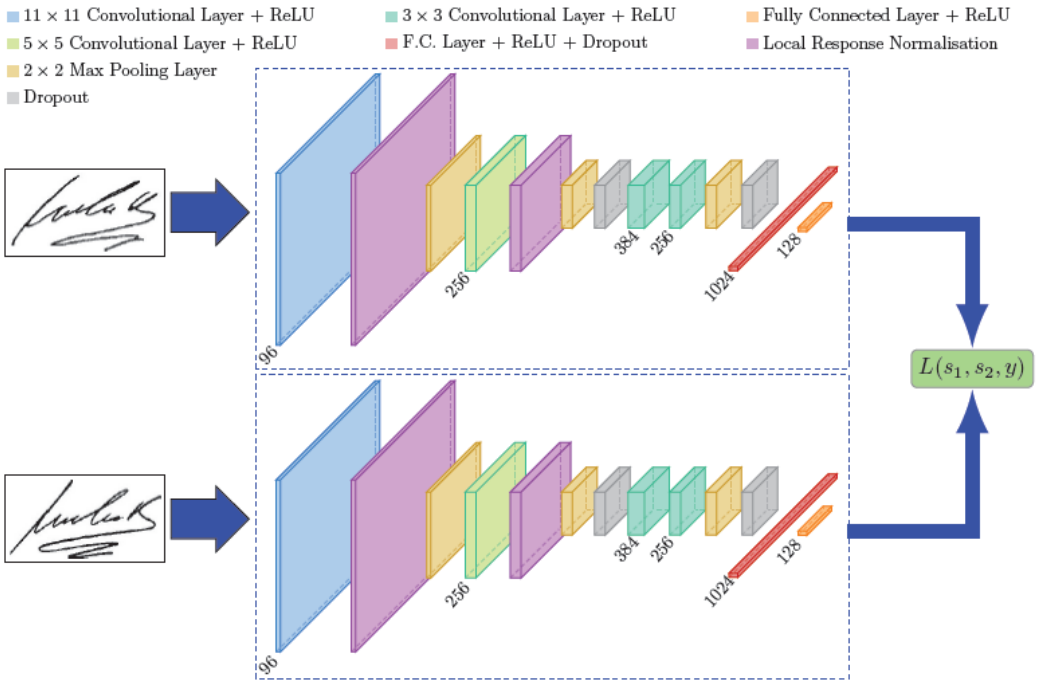


Rysunek 2.11: Przykład modelu głębokiego sieci neuronowej, gdzie 2 warstwy łączenia występują równolegle, reszta połączona jest szeregowo (standardowo), wizualizacja modelu z Tensorflow

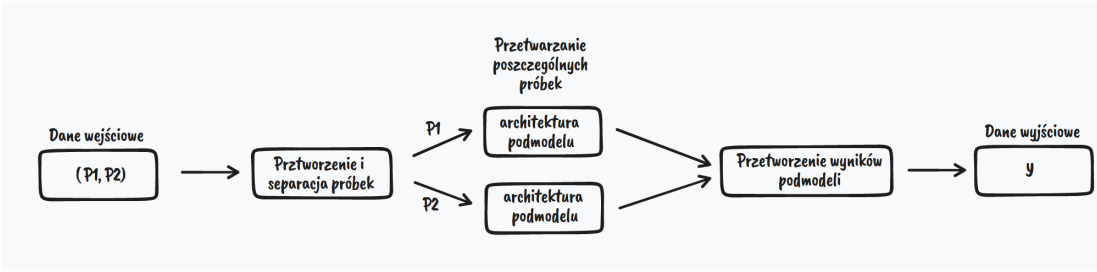
2.8.1 Model syjamski

Model syjamski [2] to koncepcja architektury modeli głębokich znajdująca zastosowania w problemach wymagających porównywania wyników próbek. Jak przedstawia rysunek 2.13 ich dane wejściowe składają się z więcej niż jednego elementu, które równolegle przetwarzane są przez takie same podmodele. Elementy wejściowe mogą być odpowiednio przetworzone, a potem odseparowane by oddzielnie stać się próbkami wejściowymi podmodeli. W praktyce zazwyczaj jest to jeden podmodel wykorzystywany więcej niż jeden raz w procesie otrzymywania wyniku z modelu (głównie dla optymalizacji, lecz użycie 2 modeli z identycznymi architekturami i wagami jest także możliwe) na podzielonej wcześniej próbce danych. Każdy element po podziale jest indywidualnie przekazywany przez podmodel, a ich rezultaty są porównywane. Jeden z elementów staje się podstawą, do którego inne chcą być podobne (lub wręcz odwrotnie jak najmniej

podobne).Wyniki modeli syjamskich to często ocena prawda/fałsz, przy porównaniu elementów do siebie, na przykład ocena autentyczności jak przykład na rysunku 2.12.



Rysunek 2.12: Przykład modelu głębokiego sieci neuronowej o architekturze modeli syjamskich [2].



Rysunek 2.13: Schemat modelu syjamskiego.

Użyteczne jest to także w przypadku trenowania modelu do klasyfikacji danych do wielu klas. Dodanie, czy usunięcie jednej z nich, to kompletnie nowy proces uczenia. Model syjamski podchodzi do sprawy od innej strony, oceniając podobieństwo próbek, w takim momencie liczba klas czy danych nie jest tak sztywna. Taką samą logikę porównywania można stosować do wyszukiwania duplikatów próbek - ich podobieństwo powinno być najwyższe.

2.8.2 Funkcja potrójnej straty

Funkcja potrójnej straty (eng. triplet loss) [4] stosowana jest w modelach pracujących na danych wejściowych złożonych ze zbiorów trzech próbek danych. Każda z nich przekazywana jest przez model, który tworzy reprezentację wektorową próbek, by potem można było porównać je do siebie. Zestaw tworzony

jest na podstawie pojedynczej próbki (nazywana kotwicą), do której dobierany zostaje przykład pozytywny - tworzą one razem parę pozytywną, oraz przykład negatywny - tworzą parę negatywną. Stworzone pary wykorzystane zostają do obliczenia dystansu pomiędzy próbkami, najczęściej Euklidesowy lub cosinusowy. Te w parze pozytywnej powinny być blisko siebie (mały dystans), natomiast pomiędzy próbkami z pary negatywnej dystans powinien być większy. Pary mogą być wybierana na podstawie przynależności do tego samego klastra (dla pary pozytywnej), czy też zupełnie innej zależności, której model ma się nauczyć. Proces uczenia ma za zadanie zmniejszenia dystansu pomiędzy elementami z par pozytywnych oraz zwiększenia dystansu pomiędzy próbkami z par negatywnych.

2.9 Metody porównywania podobieństwa wektorów

2.9.1 Odległość Euklidesowa

Odległość Euklidesowa [11] zdefiniowane dla punktów P i Q o trzech wymiarach P = [x,y,z], Q=[a,b,c] przedstawia równanie (2.16). Odpowiadające wartości wymiarów są odejmowane od siebie, a ich różnica potęgowana do drugiego stopnia. Wyniki są sumowane i wyciągany zostaje pierwiastek, czyli wpływ na wynik mają różnice w wartościach w poszczególnych wymiarach.

$$d_{Euclidean}(P,Q) = \sqrt{(x-a)^2 + (y-b)^2 + (z-c)^2} \tag{2.16}$$

2.9.2 Podobieństwo cosinusowe i odległość cosinusowa

Odległość cosinusowa to kolejna metoda określania jak bliskie sobie są próbki. Jest ona zdefiniowana na równaniu (2.17), jako różnica między jedynką a podobieństwem cosinusowym, które jest wartością z przedziału od zera do jeden. Mała wartość podobieństwa cosinusowego odpowiada dużej wartości odległości cosinusowej. Równanie 2.18 przedstawia wzór podobieństwa cosinusowego dla dwóch wektorów P i Q. $|P|$ to norma Euklidesowa wektora P, czyli licznik dzielony jest przez długości wektorów normalizując wartości do przedziału od 0 do 1. Zwracana wartość to miara cosinusa kąta pomiędzy wektorami, im większe podobieństwo wektorów (wartość bliższa 1) tym mniejszy kąt pomiędzy wektorami, zero występuje w momencie, gdy wektory są ortogonalne (90 stopni kąt w każdym z wymiarów). Odległość cosinusowa jest duża, gdy kąt pomiędzy wektorami jest duży i maleje wraz ze wzrostem podobieństwa (zmniejszeniem się kąta między wektorami).

$$d_{Cosine}(P,Q) = 1 - p_{Cosine} \tag{2.17}$$

$$p_{Cosine}(P,Q) = P * Q/|P| * |Q| \tag{2.18}$$

Część II

Część praktyczna

Rozdział 3

Implementacja

3.1 Analiza problemu

Urządzenia do transmisji sygnału telekomunikacyjnego to złożone maszyny pod względem sprzętowym jak i oprogramowania. Wykorzystywane są nieustanie, a najważniejszymi cechami jest niezawodność oraz prędkość, dlatego firmy takie jak Nokia kładą ogromny nacisk na testowanie nowych elementów, a także utrzymanie i kompatybilność starszych. Każda zmiana może powodować nieoczekiwane rezultaty, a wpływ zewnętrznych anomalii np. sprzętu telefonicznego łączącego się z urządzeniami niepożądane efekty. Zadaniem testerów jest sprawdzenie, czy nowe zmiany będą działały poprawnie, a aktualizacje usprawnią funkcjonalność urządzenia. Ogromna liczba linii kodu powoduje zapotrzebowanie na testerów liczone nie w setkach a tysiącach. Ich praca, choć podzielona na odpowiednie podzespoły może się nakładać, lub różne testy mogą powodować znalezienie tego samego błędu. Efektem tego jest większa liczba zgłoszeń, niż rzeczywistych błędów, powoduje to logistyczne problemy, a także zwiększone koszty naprawy. Jeśli we wczesnych etapach nie zostanie to wykryte, zespoły deweloperskie równolegle pracują nad naprawą tego samego. To niepożądane zjawisko jest rozpatrywane przez wyznaczone do tego osoby, lecz ludzki umysł nie jest stworzony do analizy i zapamiętania tak dużej ilości zgłoszeń, by porównać ich zawartość. Narzędzia podpowiadające możliwe powielenia mogłyby usprawnić ten proces. Rozdział implementacji poświęcony jest eksperymentom, których celem jest znalezienie odpowiedniej kombinacji metod mogących posłużyć do stworzenia takich narzędzi.

3.2 Przebieg badań

Pracę badawczą rozpoczęto od analizy zbioru danych za pomocą statystyk, jak i badania składu poszczególnych pól formularzy. Kolejnym etapem było przygotowanie wektoryzacji danych tekstowych za pomocą TFiDF oraz modelu głębokiego BERT. Obie metody zostały wykorzystane w kolejnych etapach. Po zrozumieniu zbioru i przygotowaniu odpowiedniego formatu danych, następnym etapem było wykorzystanie algorytmu PCA do oceny znaczenia poszczególnych pól formularza w kategorii wpływu na poprawność wyników. Tą samą metodę użyto do wizualizacji próbek w dwóch wymiarach. Reprezentację danych na płaszczyźnie wykonano także w kolejnych krokach za pomocą Truncated-SVD, t-SNE oraz UMAP z wyko-

rzystaniem informacji uzyskanych przy pomocy analizy pól formularza i wektoryzacją TFiDF. Algorytmy te zastosowano także do próby klasyfikacji formularzy przy wykorzystaniu podobieństwa cosinusowego. Wektory ze zredukowaną wymiarowością przez wspomniane metody wykorzystano, jako dane wejściowe dla algorytmu (dodatkowo przygotowano przypadek z użyciem wektoryzacji BERT, bez algorytmów redukcji wymiarowości) lasu losowego do wyszukiwania duplikatów. Jako alternatywną metodę wyszukiwania duplikatów w zbiorze formularzy wykorzystano model głęboki o architekturze modeli syjamskich z potrójną funkcją straty oraz danymi wejściowymi wektoryzacji BERT.

3.3 Opis zbioru danych

Zbiorem danych, na którym były prowadzone badanie jest zestawieniem wszystkich historycznych zgłoszeń odpowiednio przefiltrowanych, aby pasowały do zaplanowanego przeznaczenia. Głównym elementem odsiewającym niepotrzebną część było przynależność do komponentu deweloperskiego, na którym wykryto błąd (musiał znajdować się na liście 11, które będą obsługiwane). Ważnym elementem jest także wiek zgłoszenia, ponieważ komponenty i oprogramowanie zmienia swoje funkcjonowanie na przestrzeni lat. Każde zgłoszenie, to formularz z polami tekstowymi. Tablica 3.1 przedstawia zestawienie pól zawartych w formularzu zgłoszenie, jednak nie wszystkie z tych pól można wydajnie stosować. Elementy 'group in charge' są wypełniane po zamknięciu zgłoszenia, inne pola często są pozostawiane puste, a dokładność testerów podczas wypełniania znacząco spada wraz z każdym kolejnym wymaganym polem. Komponenty deweloperskie zawarte w 'group in charge' jest to dana kategoryczna definiująca, jakiego komponentu dotyczył błąd zaraportowany w formularzu. Zbiór danych zawiera 11 komponentów najczęściej występujących, z wystarczającą liczbą próbek by uwzględniać je w ocenach modeli. W polach, które w teorii powinny być kategoryczne, natomiast przez testerów są wpisywane, jako tekst często pojawiają się błędy zduplikowanych liter, niepotrzebnej spacji itp. W czasie analizy zbioru danych nie znalazłem jakichkolwiek silnych zależności między takiego typu danymi, a zduplikowanymi zgłoszeniami. Mają one często inne nazwy produktów, do których zastały przypisane, czy też wersje sprzętu na którym błąd został znaleziony. Każdy tester na swój własny sposób (choć zgodny z polami formularza) próbuje opisać znaleziony błąd, co nie sprzyja znajdowaniu zależności w prezentowanym zbiorze. W sekcji 3.4 opisuję sposób przetworzenia danych do użytkowej formy natomiast w sekcji 3.6 analizuję pola formularza po względem użyteczności przy kategoryzowaniu formularzy.

3.4 Opis przetwarzania zbioru danych

Dane, które trafiły do mnie przetworzone zostały przez wewnętrzne narzędzie zaprojektowane do wyciągania elementów z formularza, dodatkowo potok ETP (Extract, Transform, Process), który jest także wewnętrznym narzędziem przefiltrował dane wraz z zadanymi zależnościami. Wybierany jest odpowiedni przedział czasowy formularzy dla zbioru danych oraz kategoryczne grupy biznesowe dla których narzędzie ma być przeznaczone. Po przetworzeniu dane przekazywane są w formacie JSON, a pola podzielone są

Nazwa	Typ	Rodzaj
title	str	tekstowa
tags	str	tekstowa
state	str	tekstowa
severity	str	tekstowa
reported by	str	tekstowa
feature id	str	tekstowa
feature jira id	str	tekstowa
feature title	str	tekstowa
feature summary	str	tekstowa
reported date	str	tekstowa
closed date	str	tekstowa
state history	str	tekstowa
reason for transfer	str	tekstowa
transfer history	str	tekstowa
transfer history units	str	tekstowa
software build	str	tekstowa
sw components ref	str	tekstowa
sw components fixed	str	tekstowa
desc expected	str	tekstowa
desc actual	str	tekstowa
desc analysis	str	tekstowa
desc test steps	str	tekstowa
desc test line	str	tekstowa
desc test last sw	str	tekstowa
author	str	tekstowa
corrections	str	tekstowa
modified components	str	tekstowa
fault analysis root cause	str	tekstowa
fault analysis	str	tekstowa
fault analysis product subsystem	str	tekstowa
group in charge name initial	str	tekstowa
group in charge unit initial	str	tekstowa
group in charge raw unit	str	tekstowa
group in charge raw tribe	str	tekstowa
system component	str	kategoryczna
product name	str	kategoryczna
problem type	str	kategoryczna
recovery action	str	kategoryczna
feature release	str	kategoryczna
feature domain	str	kategoryczna
feature competence area	str	kategoryczna
feature system	str	kategoryczna
feature level	str	kategoryczna
discovered in	str	kategoryczna
software release	str	kategoryczna
author group name	str	kategoryczna
author group tribe	str	kategoryczna
author group unit	str	kategoryczna
group in charge name	str	kategoryczna
group in charge unit	str	kategoryczna
duplicates	lista	generowana

TABLICA 3.1: Tablica pól formularza zgłoszenia.

na te uznane za kategoryczne oraz tekstowe. Listing 3.1 prezentuje przykładowy formularz przekazany do analizy.

```
{'id_formularza':{
    'classical':{
        <wszystkie pola tekstowe wraz z wartosciami>
        },
    'categorical':{
        <wszystkie pola kategoryczne wraz z wartosciami>
        }
    }
}
```

Listing 3.1: Przykładowe pola zgłoszenia dla przetworzonego formularza

Jak już wspominałem tekst wprowadzony przez użytkownika bardzo często zawiera błędy, specjalne znaki itd. Listing 2.1 prezentuje proces czyszczenia tekstu dla każdego z pól formularza wykonywany w celu unifikacji i kompresji dokumentu do tych, które pomogą w rozwiązaniu problemu. Wykorzystuje ona funkcje zaprojektowane bezpośrednio do wykorzystania dla tego zbioru danych i jest używana przez zespoły do procesowania danych tekstowych wprowadzanych przez użytkownika na temat oprogramowania 5G.

Oczyszczone dane kategoryczne zamieniłem na kodowanie gorąco-jedynkowe, co jest najpopularniejszym sposobem używania tego typu danych w modelach uczenia maszynowego. Dane tekstowe, zamieniłem na wektory liczb używając dwóch sposobów i opisuję to w podsekcjach 3.4.1 oraz 3.4.2. Ten prostszy, o mniejszym zapotrzebowaniu obliczeniowym - TFiDF wykorzystałem do początkowych badań, natomiast model BERT, głęboka sieć neuronowa stworzona na podstawie warstw atencji, wykorzystuję do porównania w metodach wyszukiwania duplikatów.

3.4.1 TFiDF

Do wektoryzacji dokumentów tekstowych TFiDF wykorzystuję bibliotekę *scikit-learn* oraz klasę TfidfVectorizer. Dane zaimportowane do klasy DataFrame biblioteki *pandas* przekazuję do funkcji listing 3.2.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfIdfVect = TfidfVectorizer(use_idf=True)
tfIdf = tfIdfVect.fit_transform(df['text'])
```

Listing 3.2: Kod python do tworzenia wektoryzacji TFiDF z użyciem biblioteki *sklearn*

Kolumna 'text' zawiera wybrane pola formularza połączone w jeden ciąg znaków. W trakcie badań wykorzystywane są różne kombinacje pól, lecz w sekcji 3.6.1 rozpatrywane są pola osobno w celu oceny ich znaczenia w poprawnych predykcjach.

3.4.2 BERT

Podstawowy model BERT jest dostępny do pobrania z oficjalnego rejestru Google w różnych wielkościach i trenowany na różnych zestawach danych. W badaniach użyto 110-miliono parametrowy model, z

12 warstwami atencji, zwracający reprezentację liczbową ciągów znakowych jako wektor o długości 768 dla każdego przekazanego słowa. Limit ilości wprowadzanych tokenów to 128, a model był wytrenowany na zbiorze dokumentów w języku angielskim. Model podzielony jest na dwie warstwy tj. preprocesującą dane wejściowe oraz właściwy model, czyli część treningową by model BERT poznał zależności językowe charakterystyczne dla formularzy błędów.. Jak można zauważyć na listingu 3.3 istnieją odpowiednie funkcje by dodać je jako warstwy tworzonego modelu. Listing 3.3 oraz 3.4 prezentuje użytą architekturę, którą wykorzystałem do douczania modelu. Sam proces uczenia podzielony został na dwa etapy, a jako etykiety użyto przynależność formularzy do danych komponentów deweloperskich, czyli danej kategorycznej 'group in charge unit'. Pierwszy etap to uczenie modelu z zamkniętymi warstwami BERT, który trenuje wyłącznie dodane przeze mnie warstwy nazywane 'tuning'iem', natomiast w drugim etapie odblokowano 12 warstwę atencji BERTA'a i ponowiono trening na niższym współczynniku uczenia się 1e-5, w porównaniu do pierwszego etapu gdzie użyto 1e-3. Warto również nadmienić, że modele Google o architekturze BERT zwracają różne wersje danych wyjściowych, powszechnie używanym jest 'pooled_output', który agreguje wynik wszystkich przekazanych tokenów do jednego tj. wymiar 1x768. Tablica 3.2 prezentuje wyniki modelu.

```
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')

encoder_inputs = hub.KerasLayer('./bert_preprocess', trainable=False, name="
    bert_preprocess")(text_input)
outputs = hub.KerasLayer('./bert_model', trainable=False, name="bert_encode")(
    encoder_inputs)
net = outputs['pooled_output']
dense_layer1 = tf.keras.layers.Dense(units=768,activation='selu')(net)
drop1 = tf.keras.layers.Dropout(0.2)(dense_layer1)
dense_layer2 = tf.keras.layers.Dense(12, activation='sigmoid')(drop1)
out = dense_layer2

basic_model = tf.keras.Model(inputs=text_input, outputs=out, name='BERT_dense_out')
basic_model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(), optimizer=
    keras.optimizers.Adam(lr=1e-3))
```

Listing 3.3: Architektura modelu z warstwani BERT

```
Model: "BERT_dense_out"
__________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==================================================================================
 text (InputLayer)           [(None,)]                 0           []

 bert_preproc              {'input_word_ids':   0           ['text[0][0]']
(KerasLayer)                        (None, 128),
                            'input_type_ids':
                                (None, 128),
                            'input_mask': (Non
                                e, 128)}
```

```
bert_encode                {'pooled_output': (  109482241   ['bert_preproc[0][0]',
(KerasLayer)               None, 768),                       'bert_preproc[0][1]',
                           'sequence_output':                'bert_preproc[0][2]']
                           (None, 128, 768),
                           'default': (None,
                           768),
                           'encoder_outputs':
                           [(None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768),
                           (None, 128, 768)]}

dense (Dense)              (None, 768)           590592      ['bert_encode[0][13]']

dropout (Dropout)          (None, 768)           0           ['dense[0][0]']

dense_1 (Dense)            (None, 12)            9228        ['dropout[0][0]']

==================================================================================
Total params: 110,082,061
Trainable params: 599,820
Non-trainable params: 109,482,241
```

LISTING 3.4: Opis wymiarowości danych wychodzących z warstw dla modelu wykorzystującego warstwy BERT

Nazwa	Wynik
BERT tunned	83,92%
BERT fine tuned	84,10%

TABLICA 3.2: Tablica wyników modelu.

3.5 Walidacja modeli

Walidacja wykonywana jest na wydzielonych z całości zbiorach testowym i walidacyjnym. Podział to 10% dla testowego (4695 formularzy) oraz 20% dla walidacyjnego (9390 formularzy), pozostałe 70% to zbiór treningowy (32866 formularzy). Dodatkowym warunkiem jest to by każdy z jedenastu komponentów deweloperskich miał swoje odwzorowanie w zbiorach. Ponieważ ich liczba nie jest zbalansowana to ilość

formularzy, które powinny trafić do danego podzbioru jest wydzielana bezpośrednio dla danego komponentu. Tablica 3.3 przedstawia nierówne rozłożenie formularzy w zbiorze danych.

Kategoria	Liczba formularzy w zbiorze
1	4436
2	3220
3	2657
4	5128
5	3668
6	1161
7	4027
8	19938
9	2228
10	266
11	223

TABLICA 3.3: Liczba formularzy w zbiorze należąca do danej kategorii.

3.6 Klastrowanie danych z użyciem PCA

Narzędzie PCA użyto w pierwszej kolejności do oceny pól formularzy pod względem użyteczności, by następnie sprawdzić, czy metoda daje poprawne wyniki w rzeczywistym wyszukiwaniu duplikatów.

3.6.1 Analiza pól formularza z wykorzystaniem PCA oraz podobieństwa cosinusowego

Na początek oceniona została wydajność pól w kategoryzacji danych do komponentów deweloperskich, co jest uproszczeniem problemu (formularze zduplikowane muszą należeć do tego samego komponentu). Za pomocą wektoryzacji danych tekstowych metodą TFiDF, zamieniono poszczególne pola formularzy zbioru treningowego na wektory i zmniejszono ich wymiarowość za pomocą algorytmu PCA. Ostatecznie wyniki modelu zostały ocenione za pomocą podobieństwa cosinusowego, sprawdzając czy formularz o największym podobieństwie jest z tego samego komponentu deweloperskiego. Listing 3.5 przedstawia funkcję użytą do porównania podobieństwa formularzy.

```
from sklearn.metrics import pairwise
from tqdm import tqdm
import numpy as np

def cos_sim_by_du(fitted_tool, fitted_tool_test, labels_train, labels_test):
    count={ el:0 for el in range(11)}
    sim_count={ el:0 for el in range(11)}

    for i in tqdm(range(len(fitted_tool_test))):
        sim = pairwise.cosine_similarity(fitted_tool, Y=[fitted_tool_test[i]])
        if labels_train[np.argmax(sim)] == labels_test[i]:
            sim_count[int(labels_test[i])] +=1
        count[int(labels_test[i])] +=1

```

```
15      for el in range(12):
16          print(f'Found: {sim_count[el]}/{count[el]}    {round(sim_count[el]/count[el] *
        100,2)}% ')
17
18      print("RESULT")
19      print(f'{sum([v for v in sim_count.values()])} / {sum([v for v in count.values()])}'
        )
```

Listing 3.5: Kod python do porównania podobieństw cosinusowych formularzy

3.6.2 Wizualizacja danych z analizy PCA

Zmniejszenie wymiarowości wektorów TFiDF dla danych do dwóch komponentów pozwoliło na prezentację ich klastrów na płaszczyźnie. Zestaw danych przetworzono w dwóch wariantach. Pierwszy wykorzystywał wyłącznie pole 'title' do tworzenia wektorów, natomiast drugi łączył w sobie trzy pola o najwyższej skuteczności w eksperymencie z tablicy 3.4.

Rysunek 3.1 przedstawiający reprezentację graficzną dla tytułów nie prezentuje żadnych zależności między przynależnością do komponentów, a miejscem na wykresie. Dane nie przedstawiają widocznych różnic. Rysunek 3.2 natomiast prezentuje dwa klastry (przy 11 pożądanych), co pokazuje, że użycie większej ilości pól formularza pozytywnie wpływa na klasteryzację. Jeden z klastrów zawiera dwa komponenty deweloperskie, których formularze na tyle różnią się od innych by ich reprezentacja różniła się znacząco od reszty. Osi na obu wykresach odpowiadają wartościom próbek w danym indeksie wektora (długości wektorów zostały zredukowane do 2).

3.6.3 Walidacja

Próbki ze zbioru testowego przyrównywane są po kolei do wszystkich próbek ze zbioru treningowego. Wyliczane są dystanse cosinusowe, by wyszukać próbkę z największym podobieństwem do tej testowej. Następnie sprawdzane jest jak często ich komponenty deweloperskie są takie same, czy oba dokumenty należą do tego samego klastra.

W tabeli 3.4 zgromadzono wyniki dla poszczególnych pól pokazując liczbę par z pasującymi do siebie komponentami w kolumnie 'Wynik', liczbę wszystkich próbek które posiadały wypełnione dane pole w 'Wszystkie' oraz procentowe przedstawienie wyniku pozytywnego dopasowania komponentów deweloperskich w kolumnie 'Procent'. Wyniki oscylują w przedziale pomiędzy 18, a 37 procent, co wskazuje, że każde pole potrafi pozytywnie wpłynąć na wyniki (w losowym przypisywaniu komponentów było by 1 na 11 szansy na trafienie, co odpowiada około 9% trafności), lecz nie potrafią ostatecznie rozwiązać problemu przypisania komponentów.

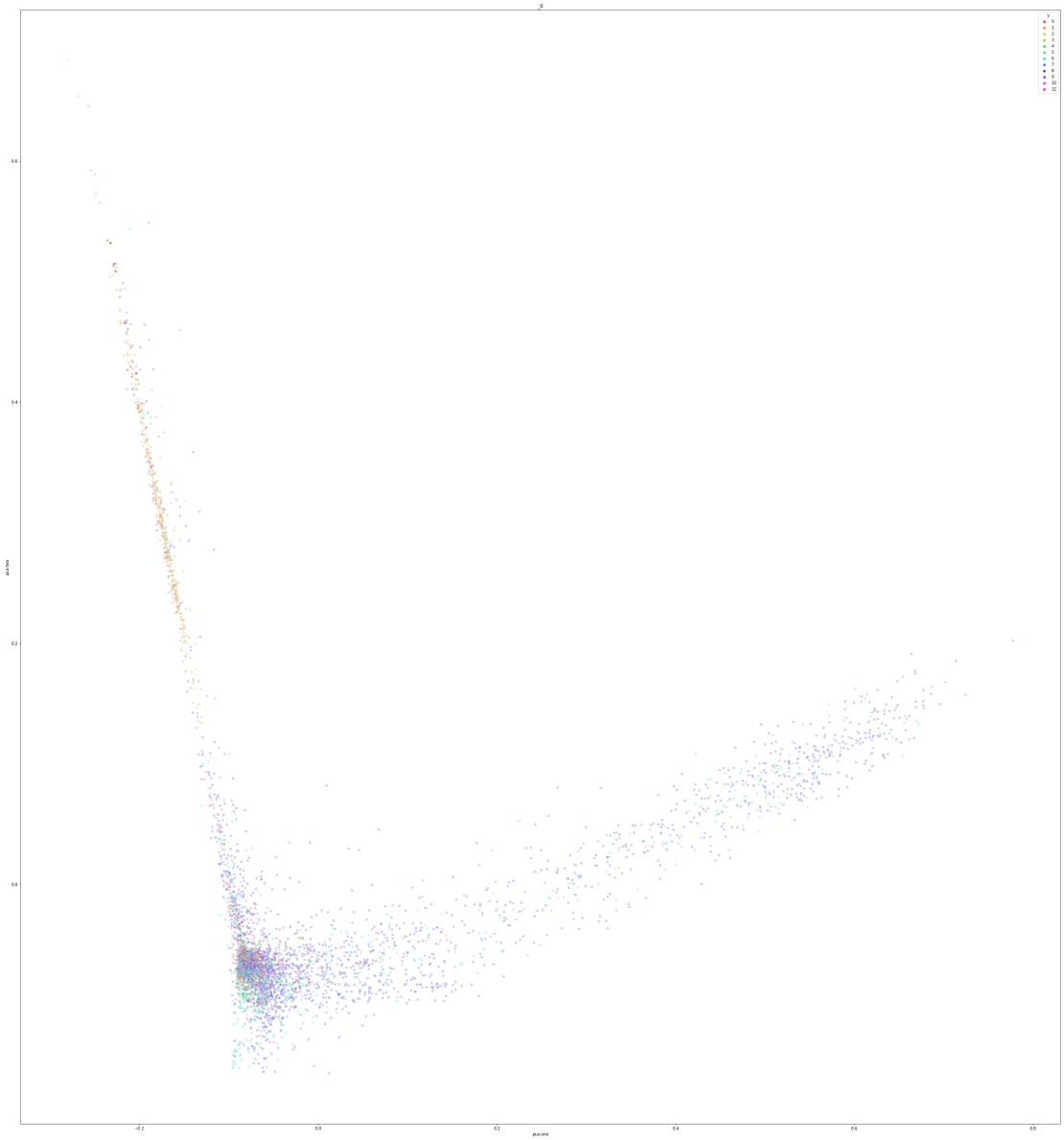
3.7 Klastrowanie danych z użyciem Truncated-SVD

Użycie Truncated-SVD, posłużyło do porównania wyników z poprzednio używaną PCA i potwierdzenia, że algorytmy liniowej redukcji wymiarowości danych nie radzą sobie z uproszczoną wersją problemu, czyli

Rysunek 3.1: Przedstawienie wektorów TFiDF dla tytułów formularzy skompresowane do dwóch wymiarów przy użyciu algorytmu PCA.

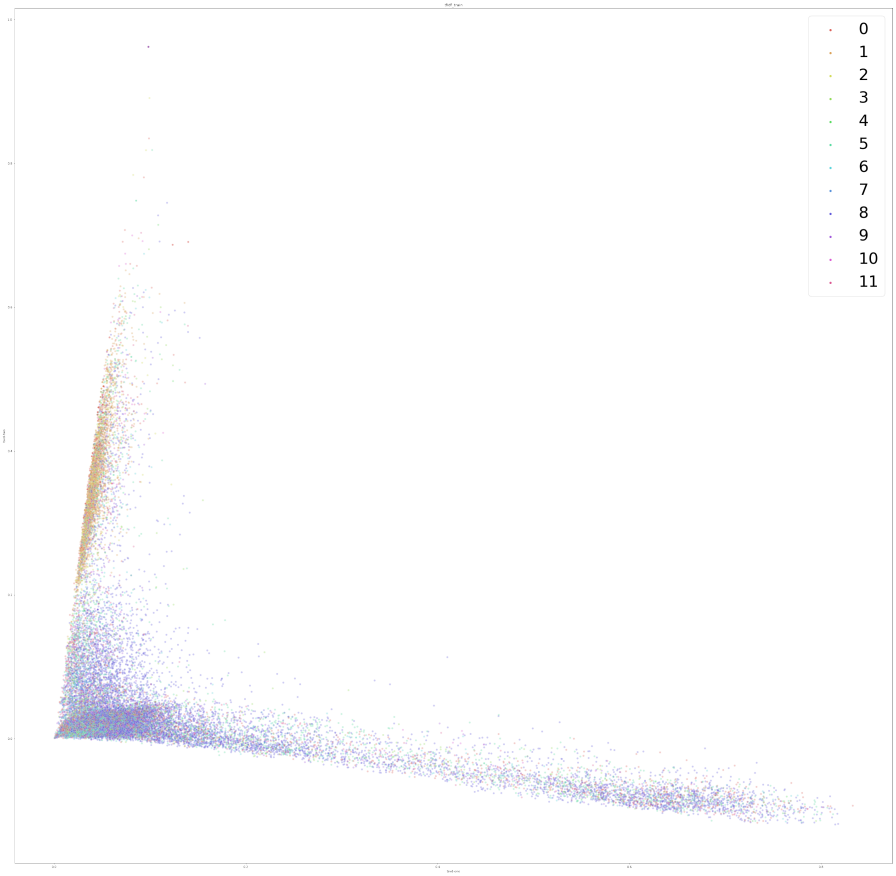
Nazwa pola	Wynik	Wszystkie	Procent
Tile	1810	4869	37,17%
Tags	1398	4869	28,71%
Feature title	890	4671	18,28%
Feature summary	944	4671	19,38%
Expected description	1557	4869	31,98%
Actual description	1798	4869	36,93%
Analysis desription	1814	4869	37,26%
Test steps description	1546	4869	31,75%
Test line description	1219	4869	25,04%
Test last software description	1012	4718	20,78%

Tablica 3.4: Tablica wyników dla poszczególnych pól.



RYSUNEK 3.2: Przedstawienie wektorów TFiDF dla tytułu, właściwego opisu oraz opisu analizy formularzy skompresowane do dwóch wymiarów przy użyciu algorytmu PCA.

klasteryzacją danych. Użyto tego samego zbioru danych, który dawał lepsze wyniki w sekcji 3.6, czyli zawierającego trzy pola o najwyższej skuteczności w eksperymencie z tablicy 3.4. Wyniki zawarto w tabeli 3.5, gdzie metody PCA, Truncated-SVD oraz opisana w sekcji 3.9 UMAP porównywane są w zależności do ilu wymiarów zostały zredukowane. Kolumna 'Wynik' prezentuje liczbę poprawnych predykcji, 'Wszystkie' liczbę wszystkich prób, a 'Components' wartość parametru określającego liczbę wymiarów, do których były zredukowane dane przed wyszukaniem najbardziej podobnych próbek. Rysunek 3.3 przedstawia rzutowanie wektorów TFiDF, próbek złożonych z tytułu, właściwego opisu oraz opisu analizy skompresowania do dwóch wymiarów (gdzie oś odpowiada wymiarowi wektora danych) przy użyciu Truncated-SVD.



Rysunek 3.3: Przedstawienie wektorów TFiDF dla tytułu, właściwego opisu oraz opisu analizy formularzy skompresowane do dwóch wymiarów za pomocą Truncated-SVD.

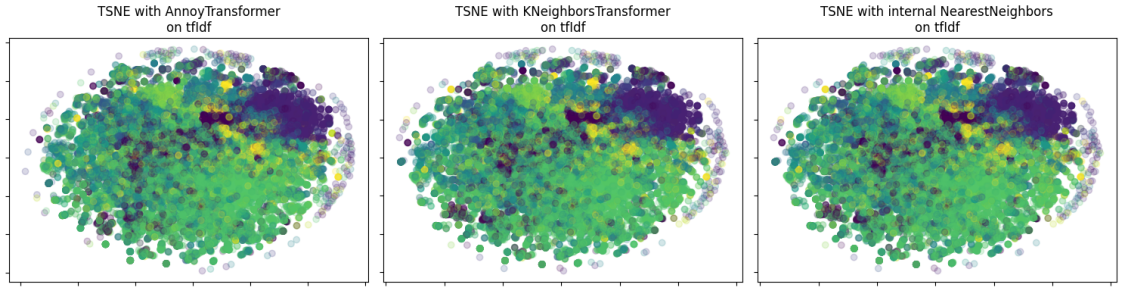
Nazwa metody	Wynik	Wszystkie	Components
PCA	2176	4869	100
PCA	2256	4869	1000
Truncated-SVD	2237	4869	100
Truncated-SVD	2266	4869	1000
UMAP	1333	4869	2
UMAP	2004	4869	100
UMAP	1938	4869	1000

Tablica 3.5: Wyniki kategoryzacji danych przy pomocy liniowej redukcji wymiarowości oraz współczynnika podobieństwa cosinusowego.

3.8 Klastrowanie danych z użyciem t-SNE

Porzucając liniowe metody redukcji wymiarowości, kolejnym etapem było przyjrzenie się nieliniowym algorytmom. Jako pierwsza została podjęta t-SNE, czyli metoda najczęściej używana do wizualizacji zbioru danych na dwóch, lub trzech wymiarach. I tym razem do tego posłużyła, a celem była ocena pojawienia się

klastrów reprezentujących komponenty deweloperskie.



Rysunek 3.4: Rzutowanie próbek danych ze zredukowaną wymiarowością przez algorytm t-SNE i wykorzystaniu różnych algorytmów transformacji.

Rysunek 3.4 prezentuje użycie t-SNE z trzema algorytmami do transformacji: Annoy Transformer, K-najbliższych sąsiadów oraz wewnętrznie wbudowanej transformacji najbliższych sąsiadów w implementacji sci-kit learn metody t-SNE. Wszystkie transformacje używają tej samej metryki kwadratowej Euklidesowej, a współczynnik nieokreśloności(ang. perplexity) został ustawiony na 300. Osie na wykresach odpowiadają wymiarom w wektorze reprezentacji. Przy każdej metodzie algorytm t-SNE wykonuje 200 iteracji, a liczba branych pod uwagę sąsiadów ustalona została za pomocą formuły (3.1), czyli ostateczna wartość to 904.

$$cov(X,Y) = E(X * Y) - [E(X) * E(Y)] \tag{3.1}$$

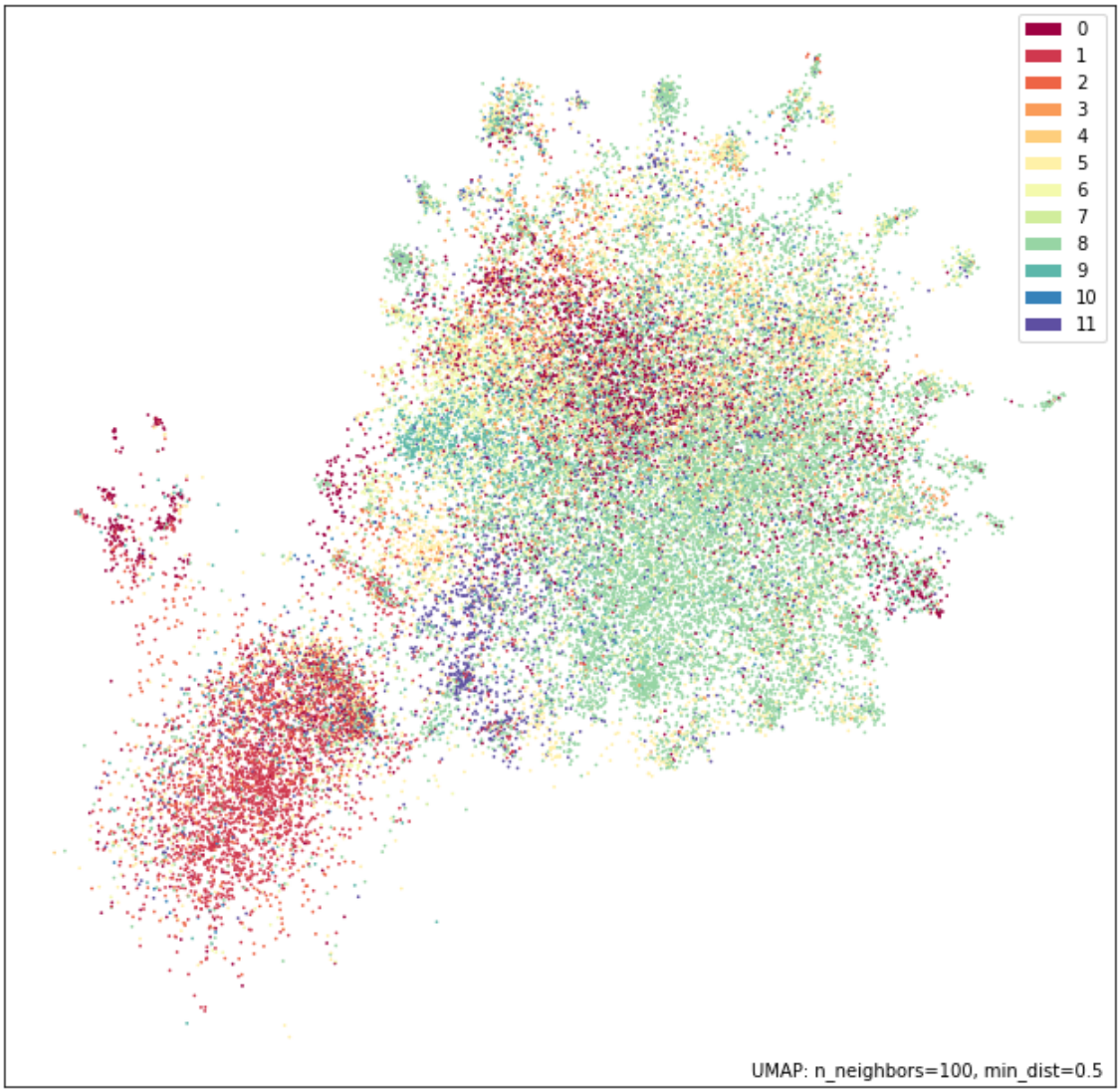
3.9 Klastrowanie danych z użyciem UMAP

Implementacja metody redukcji wymiarów UMAP pochodzi z dedykowanej biblioteki python o tej samej nazwie [5]. Przekazane wektory TFiDF reprezentujące dane o tych samych trzech polach formularza: tytuł, właściwy opis oraz opis analizy przekazano do metody fit_transform obiektu klasy UMAP. Dodatkowo zaprezentowano wyniki na zbiorze danych o polach formularza: tytuł, opis kroków testowania, właściwy opis oraz opis test-linii, co jeszcze klarowniej prezentuje wyniki klasteryzacji.

```
1
2  reducer = UMAP(n_neighbors=100,
3                 n_components=2,
4                 metric='euclidean',
5                 n_epochs=100,
6                 learning_rate=1.0,
7                 tqdm_kwds = {},
8                 verbose=True,
9                 min_dist = 0.5,
10               )
11 umap_train = reducer.fit_transform(tfIdf)
12 umap_test = reducer.transform(tfIdf_test)
```

Listing 3.6: Kod python do redukcji wymiarowości wektorów TFiDF

Rysunek 3.5 oraz 3.6 przedstawia efekt zmniejszenia ilości wymiarów wektorów TFiDF do 2 przy użyciu algorytmu UMAP oraz rzutowanie ich na płaszczyznę. 3.5 wykorzystuje dane złożone z czterech

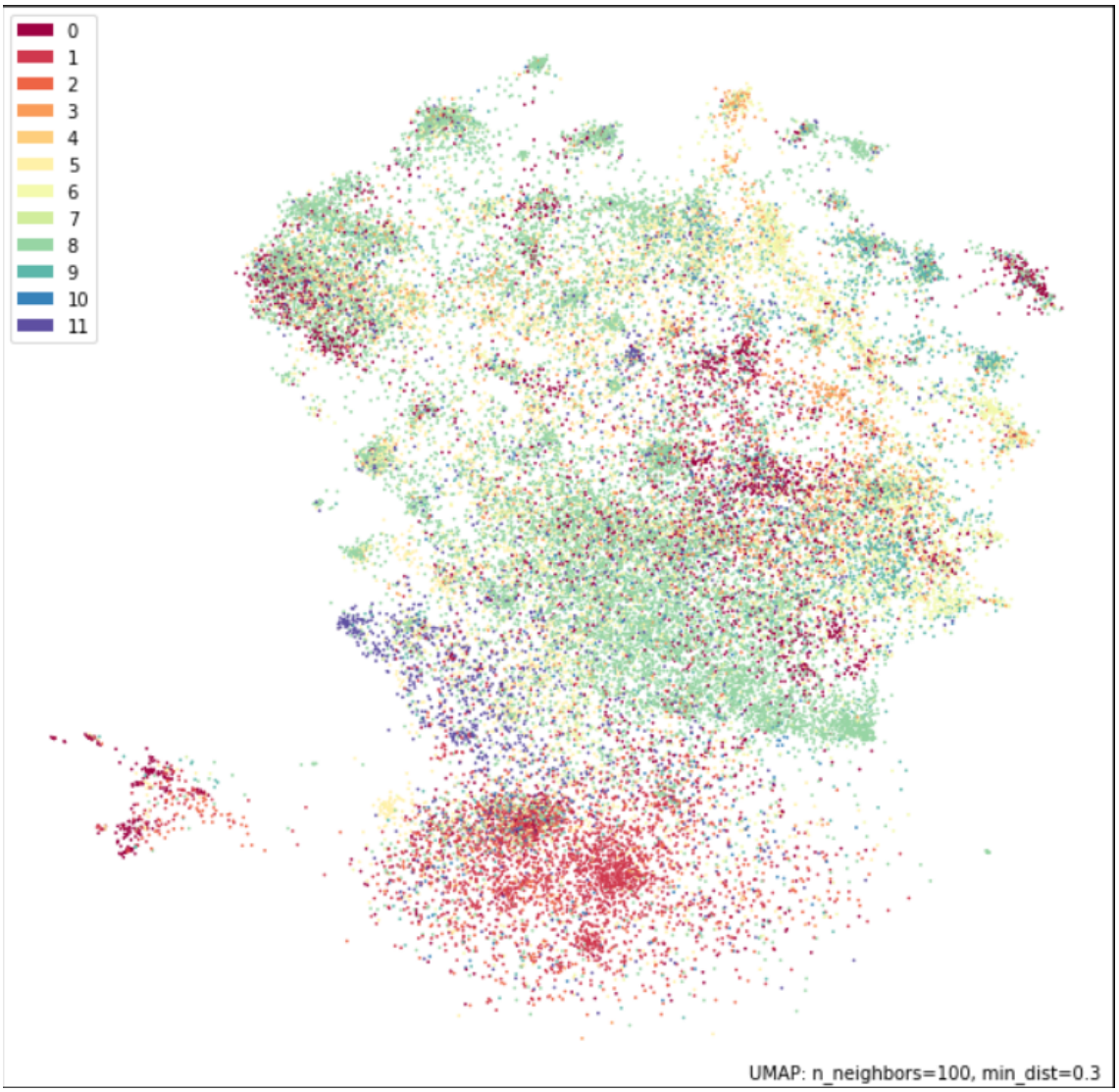


Rysunek 3.5: Reprezentacja graficzna zbioru danych z wektoryzacją TFiDF oraz redukcją wymiarowości (do 2) UMAP dla tytułu, opisu kroków testowania, właściwego opisu oraz opisu test-linii.

pól formularza: tytuł, opis kroków testowania, właściwy opis, opis test-linii, a 3.6 tytuł, właściwy opis oraz opis analizy. Listing 3.6 prezentuje użyte parametry algorytmu, w drugim przypadku min_dist ustawiono na 0.3 dla lepszej wizualizacji. Klastry komponentów deweloperskich nie zarysowują się na tyle mocno by można było wydzielić 11, lecz ich granice są bardziej widoczne niż prezentowane wcześniej metody liniowe i t-SNE. Dwie główne grupy są oddzielone znacząco (podobnie do wcześniejszych metod), dodatkowo jednak wewnątrz większej grupy można wyznaczyć mniejsze rejony z dużą przewagą formularzy z jednej grupy.

3.10 Las losowy

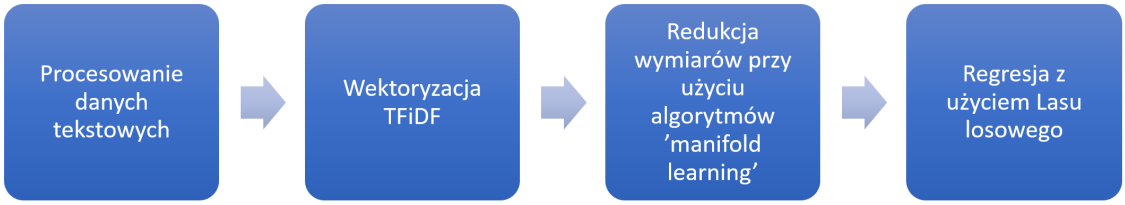
Algorytmy, które używane zostały wcześniej służą do zmniejszania wymiarowości danych. Dopiero obliczane podobieństwo cosinus'owe pozwala określić, czy dane dokumenty mają semantyczne podobieństwa. Pary formularzy porównane do siebie zwracają współczynnik, jednak każdy element dokumentu jest tak



Rysunek 3.6: Reprezentacja graficzna zbioru danych z wektoryzacją TFiDF oraz redukcją wymiarowości (do 2) UMAP dla tytułu, właściwego opisu oraz opisu analizy.

samo znaczący. Częstotliwość słów o małym znaczeniu jest traktowane, jak te kluczowe. Dodanie kolejnego etapu pozwalającego na zwrócenie uwagi na poszczególne części wektorów i poznanie właściwej semantyki tekstu jest kluczowe w kontekście znajdowania duplikatów. Podobne formułowanie zdań, czy użycie danych przymiotników, może być cechą charakterystyczną danego testera, który wypełnił formularz, lecz kolejna osoba tworząca formularz opisujący ten sam błąd opisuje go swoimi słowami, dlatego tak ważne w postawionym problemie jest określenie podobieństwa nie na podstawie użytego słownictwa (chociaż słowa kluczowe dla danych komponentów też mogą mieć duże znaczenie), lecz semantyki tekstu. Jako pierwszy algorytm został użyty las losowy w wersji regresora, z biblioteki sklearn, a cały proces został przedstawiony na wykresie 3.7. Do dopasowania drzew w lesie losowym, tak by jako wynik zwracały prawdopodobieństwo, że przekazane formularze są duplikatami, potrzebna jest zmiana danych wejściowych. Przekazywane w tym przypadku będą pary zduplikowanych formularzy (oznaczone w kolumnie 'duplicates' - tabela 3.1 - znajdują się odnośniki do duplikatów), które powinny być klasyfikowane jako 0 w skali prawdopodobieństwa oraz pary negatywne (formularze nie oznaczone jako duplikaty), a ich wyniki powinny zwracać 1. W takiej

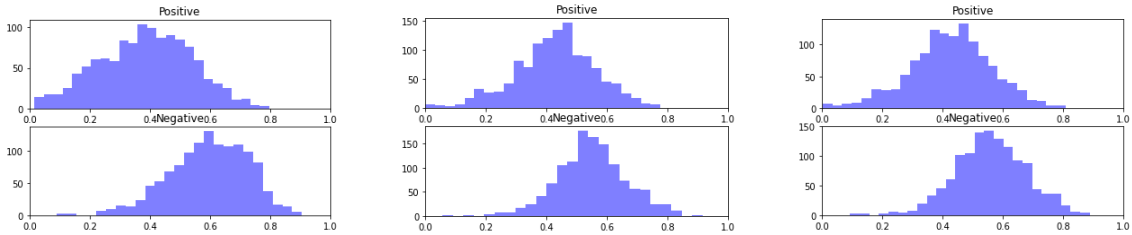
konfiguracji model predykuje dystans cosinusowy danych próbek. Wykresy przedstawione na rysunku 3.8 przedstawiają porównanie wyników przy użyciu różnych sposobów zmniejszania wymiarowości danych zaprezentowanych w poprzednich sekcjach. Średni dystans dla pozytywnych oraz negatywnych par formularzy zaprezentowany został w tabeli 3.6. Obiekt lasu losowego w każdym przypadku został zainicjalizowany tak samo jak przedstawia Listing 3.7, natomiast dane wejściowe w zależności od algorytmu były zmienianie w trakcie badania. Ostateczne, zaprezentowane wyniki na rys. 3.8 prezentują na środkowym wykresie wynik estymatora dla danych wejściowych zmniejszonych przy pomocy algorytmu Truncated-SVD do 20 wymiarów, PCA którego wynik zastosowania prezentuje wykres po prawej stronie oraz UMAP na wykresie po lewej zwraca taką samą liczbę generowanych wymiarów - 20. Ciekawym zjawiskiem jest zwężenie rozkładu dystansu wraz ze zwiększeniem ilości przekazywanych wymiarów. Średnie dystanse dla PCA i 1000 wymiarów oscylowały w okolicach 0.5 dla obu zbiorów. Estymator wybranej wielkości nie był w stanie poradzić sobie z tak złożonymi danymi. Wersję regresyjną lasu losowego wykorzystano, także wraz z wektorami utworzonymi z wykorzystaniem modelu BERT, wykres rozkładu podobieństwa przedstawia rysunek 3.9, a jego wyniki prezentowane są w ostatecznym porównaniu z modelami głębokimi w tabeli 3.7.



Rysunek 3.7: Proces przetwarzania danych do predykcji dystansu podobieństwa pomiędzy formularzami.

```
1  from sklearn.ensemble import RandomForestRegressor
2
3  RandomForestRegressor(max\_features=\'sqrt\', n\_estimators=1000, n\_jobs=20)
```

Listing 3.7: Stworzenie obiektu biblioteki sklearn dla algorytmu lasu losowego (wersja Regresora)



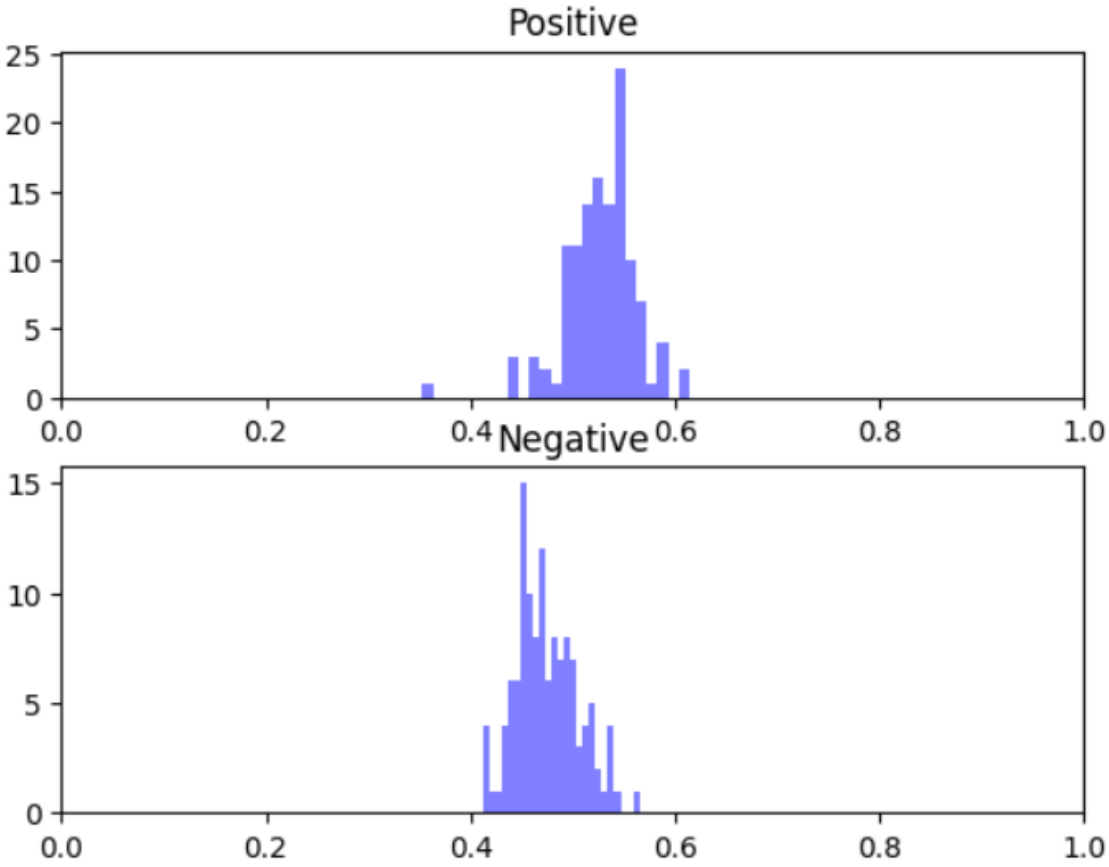
Rysunek 3.8: Rozkład dystansu przewidywany przez las losowy dla pozytywnych (duplikatów) par oraz negatywnych. Od lewej użyto UMAP, Truncated-SVD, PCA.

3.11 Model syjamski z funkcją potrójnej straty

Kolejnym etapem analizy było użycie głębokich sieci neuronowych z wykorzystaniem wektoryzacji BERT, odchodząc tym samym od TFiDF (wektor o długości około 10 000) oraz algorytmów wymaganych do

Metoda	Średni prawdopodobieństwo próbki pozytywnej	Średnie prawdopodobieństwo próbki negatywnej
PCA	58%	45%
Trunkated-SVD	55%	47%
UMAP	62%	41%

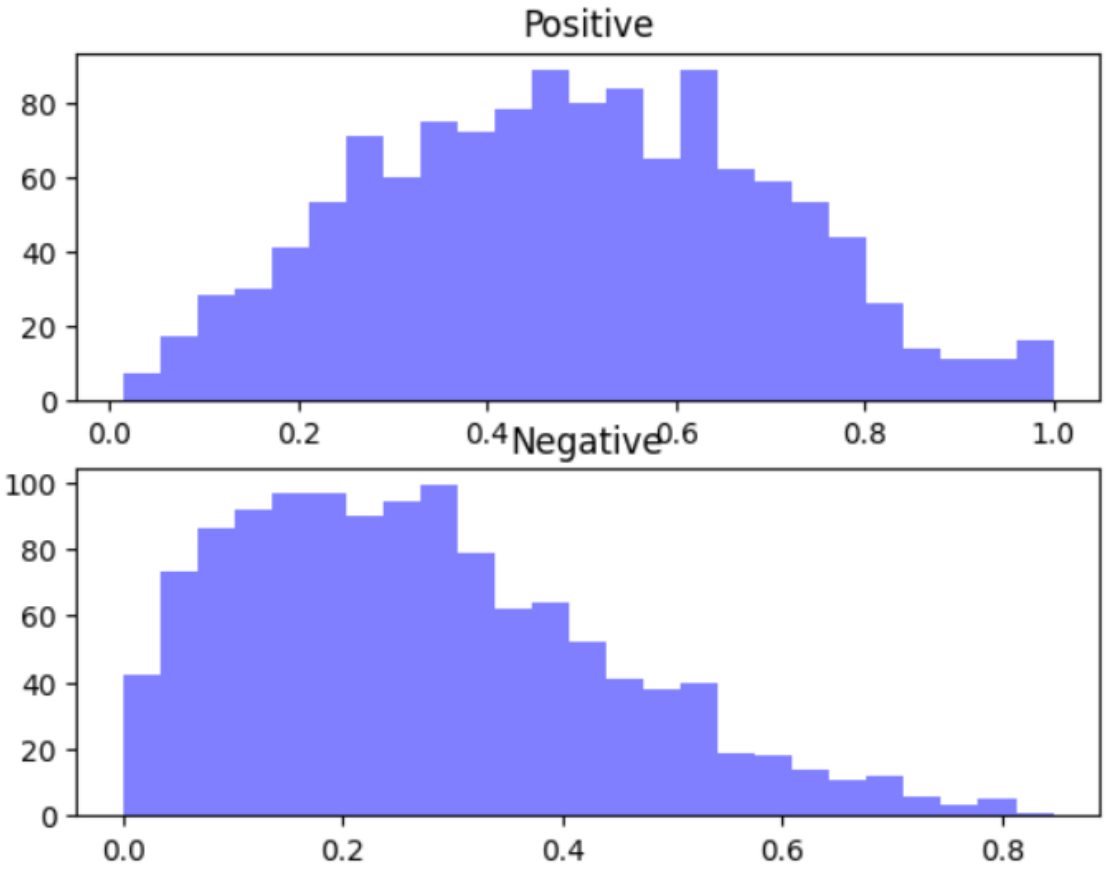
Tablica 3.6: Średnie podobieństwo cosinus'owy dla zbioru testowego próbek duplikatów (positive) oraz różnych formularzy (negative) dla lasu losowego z wykorzystaniem wektorów TFiDF i algorytmu zmniejszenia wymiarowości.



Rysunek 3.9: Rozkład dystansu przewidywany przez model lasu losowego dla pozytywnych (duplikatów) par oraz negatywnych na podstawie prawdopodobieństwa cosinus'owego.

zmniejszenia jego wymiarowości. Potrójna funkcja straty, która oblicza dystans dla danej próbki danych z jej duplikatem oraz z próbką niebędącą duplikatem wymaga, aby dane wejściowe zawierały wektory połączone w trojacze zestawienia. Podobnie jak w przypadku Lasu losowego tworzone są pary pozytywne (duplikaty), lecz tym razem, jako trzeci wektor dodany zostaje wektor niebędący duplikatem. Każdy z wektorów osobno jest przekazywany do modelu a funkcja straty porównuje wyniki, jakie model zwraca dla konkretnych próbek (ideą jest zbliżenie wyników dla duplikatów, a oddalenie dla negatywnego zestawienia) w trojaczych zestawieniach. Sama architektura składa się z warstw wektoryzacji BERT'a opisanej w sekcji 3.4.2 oraz pojedynczej warstwie z 128 neuronami. Z wektorów o tym wymiarze obliczana jest odległość Euklidesowa próbek (próbka podstawowa - duplikat oraz próbka podstawowa - próbka niebędąca duplikatem), a jej wyniki przekazywane są do funkcji straty. Podobnie jak opisano w sekcji 3.4.2, uczenie podzielone jest na etap 'tunowania' oraz 'fine-tunowania', a po pierwszej części zostaje otwarta 12-sta warstwa o architekturze

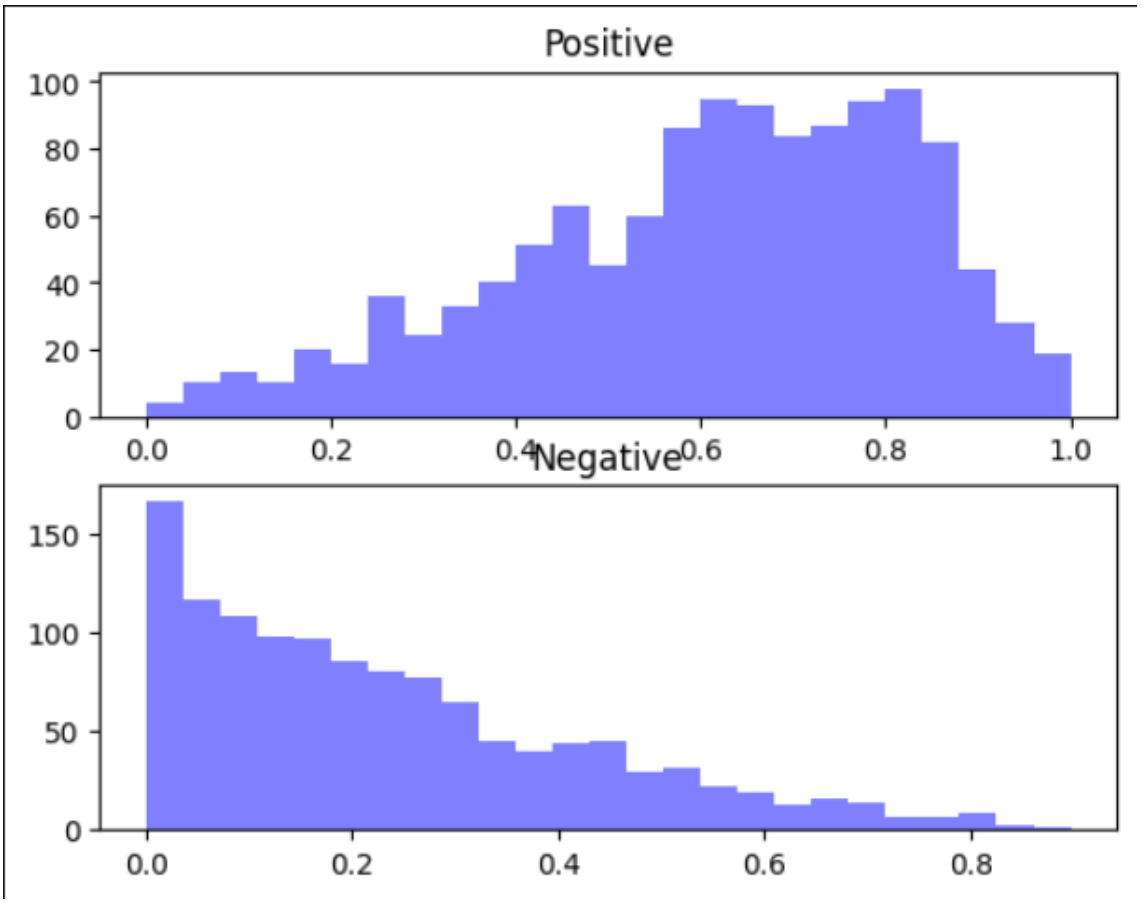
transformera w modelu BERT'a. Rysunek 3.10 przedstawia rozkład prawdopodobieństwa przy użyciu pola tytuł do predykcji wyników, model ma problemy z klasyfikacją par pozytywnych, potrafiąc w większym stopniu, gdy pary nie są duplikatami. Rysunek 3.11 przedstawia rozkład prawdopodobieństwa przy użyciu pól tytuł, opis kroków, właściwy opis, opis test linii, a jego wyniki są bardziej zbalansowane. Model potrafi ocenić zarówno czy próbka jest duplikatem oraz czy nim nie jest.



Rysunek 3.10: Rozkład podobieństwa przewidywany przez model głęboki z trojaczą funkcją straty dla pozytywnych (duplikatów) par oraz negatywnych na podstawie prawdopodobieństwa cosinus'owego z wykorzystaniem pola tytuł.

3.12 Walidacja

Porównanie wyników przedstawiono w tabeli 3.7. Użyto sposobu najbardziej obiecującego w analizie wektorów TFiDF - lasu losowego i algorytmu UMAP oraz model syjamski z wektoryzacją BERT. Ponadto wykorzystano 2 zbiory danych, w pierwszym próbka dla formularza składała się z jego tytułu, w drugim z tytułu, opisu kroków, właściwego opisu i opisu test linii. Ostatecznie można zauważyć dużą przewagę w użyciu zbiorów danych z większą ilością pól formularza, zarówno prawdopodobieństwo próbek pozytywnych jest średnio większe jak i podobieństwo negatywnych jest mniejsze. Modele syjamskie jak i las losowy w użyciem UMAP podobnie klasyfikują próbki pozytywne w tej konfiguracji, lecz określenie próbek niebędących duplikatami przychodzi sieciom głębokim z większą dokładnością. Wykorzystanie wektoryzacji BERT z lasem losowym nie przynosi pożądanych rezultatów.



Rysunek 3.11: Rozkład podobieństwa przewidywanego przez model głęboki z trojaczą funkcją straty dla pozytywnych (duplikatów) par oraz negatywnych na podstawie prawdopodobieństwa cosinus'owego z wykorzystaniem pól: tytuł, opis kroków, właściwy opis, opis test linii.

3.13 Dyskusja

Wyszukiwanie duplikatów w formularzach wypełnianych danymi tekstowymi jest trudnym zadaniem. Samo semantycznie zrozumienie tekstu nie wystarcza by poprawnie klasyfikować pary duplikatów, gdy wielu testerów w różny sposób i innymi metodami próbuje opisać ten sam problem. Stosując inne testy i różne metody, testerzy potrafią znaleźć niespójności w działaniu, które często u podnóża mają ten sam błąd. Oni sami nie wiedząc, co jest problemem opisują dokładnie to, co widzą. Wpływ na wyniki ma dokładność i sposób, z jaką tester opisuje problem, używane w danym zespole potoczne skrótowce, czy też komponent, którego jest celem przetestowanie. Wyniki pokazują, że użycie większej ilości pół formularza wpływa pozytywnie na predykcje, lecz niektóre z nich mogą jedynie zaszumić dane. Staje się zauważalne, że kolejne rubryki są wypełniane przez testerów z coraz mniejszą dokładnością, czasem nawet pomijane. Tytuł i opis są podstawowymi polami dającymi podstawę do budowy odpowiedniego zbioru. Chociaż słownictwo i zwroty powtarzają się często nawet, jeśli problemy nie są duplikatami, to zauważalna jest w wynikach tendencja klasyfikacji formularzy w odpowiedni sposób. Rozkłady negatywnych i pozytywnych próbek nachodzą na siebie, ale semantyka duplikatów pozwala na określenie ich dystansu, jako mniejszy niż średnio. Same algorytmy do redukcji wymiarowości nie wystarczają do odpowiednich predykcji, wykorzystanie ich w parze z lasem losowym, zwraca bardziej obiecujące rezultaty. Spośród badanych algorytmów to UMAP okazał

Metoda	Algorytm	Średnie prawdopodobieństwo próbki pozytywnej	Średnie prawdopodobieństwo próbki negatywnej	Pola tekstowe
TFiDF+UMAP	Las losowy	63%	41%	Tytuł, opis kroków, właściwy opis, opis test linii
BERT	Las losowy	53%	47%	Tytuł
BERT	Las losowy	53%	47%	Tytuł, opis kroków, właściwy opis, opis test linii
BERT	Model syjamski	51%	24%	Tytuł
BERT	Model syjamski	61%	23%	Tytuł, opis kroków, właściwy opis, opis test linii

Tablica 3.7: Średnie podobieństwo cosinus'owy dla zbioru testowego próbek duplikatów (positive) oraz różnych formularzy (negative)

się prezentować najlepsze wyniki w użyciu z lasem losowym. Zwrócił średnie podobieństwo na poziomie 0.63(pożądana wartość bliska 1) dla pozytywnych próbek oraz 0.41 (pożądana wartość bliska 0) średniego podobieństwa dla próbek negatywnych. Potwierdza to obserwacje z wygenerowanych poprzednio wykresów po redukcji wymiarowości, gdzie algorytm UMAP najlepiej klastrowały zbiór. Używając zaawansowanego modelu głębokiego celem było zwrócenie uwagę na szczegóły charakterystyczne dla zbioru danych, które prostsze metody nie potrafią zauważyć. Wyniki dla poszczególnego pola formularza nie różniły się znacznie od wcześniej wykorzystanego lasu losowego, lecz przy użyciu większej ilości pół - bardziej złożonych ciągów tekstowych prostsze metody pozostają w tyle za architekturami głębokimi. Dodanie większej ilości danych w tym przypadku znacznie poprawiło wyniki, co pokazuje potęgę tego modelu w kontekście pracy z bardziej złożonymi danymi wejściowymi. Jak można zauważyć w tabeli 3.7, najlepszy wynik ocenia średnio dystans podobieństwo w stosunku 0.77 do 0.39 dla próbek negatywnych i pozytywnych, co nadal pozostawia pewien stopień niepewności w ocenie duplikatów. Na rysunku 3.11 można zaobserwować, że rozkłady nadal nakładają się na siebie, więc wyodrębnienie jednej próbki z całego zestawu nadal pozostaje nie lada wyzwaniem.

Narzędzie jednak musi prezentować najbardziej prawdopodobne duplikaty osobie wyspecjalizowanej w ich ocenę. Skończona lista propozycji powinna być stworzona z wykorzystaniem uczenia maszynowego, lecz użycie podstawowych metod sztucznej inteligencji oraz statystyki do najbardziej optymalnego filtrowania oraz sortowania listy może mieć pozytywny wpływ na wyniki. Nie wszystkie pola formularzy są odpowiednie do wykorzystania w konwencjonalny sposób w modelach, lecz listy słów kluczy, czy też dane kategoryczne można stosować w procesach oceny znaczenia formularza. Nie znaleziono żadnych twardych zależności między przynależnością do jakichkolwiek pól danych kategorycznych, a znajdowaniem duplikatów, lecz zależności prawdopodobieństwa mogą pomóc w przesunięciu danego formularza wyżej na liście sugerowanych propozycji.

Rozdział 4

Wnioski

Głównym celem pracy było zaproponowaniem sposobu wyszukiwania duplikatów w formularzach zgłaszania błędów oprogramowania z wykorzystaniem uczenia maszynowego. Analizując wektoryzacje TFiDF oraz BERT, metody zmniejszania wymiarowości danych PCA, Truncated-SVD, t-SNE oraz UMAP, a także modele regresyjne las losowy oraz model syjamski z funkcją potrójnej straty osiągnięto cel proponując rzeczywiste rozwiązanie problemu. Wybrano pola formularza prowadzące do najlepszych wykrywania duplikatów: tytuł, opis kroków, właściwy opis oraz opis test linii. Na podstawie wyników można zaobserwować, że model syjamski wykorzystujący wektoryzację BERT'a oraz funkcję potrójnej straty zwraca najlepsze rezultaty podobieństwa cosinusowego par formularzy.

Uzyskane wyniki pokazują wyższość modeli głębokich w pracy z bardziej złożonymi danymi i możliwość ich zastosowania do wyszukiwania semantycznych podobieństw w formularzach.

Powstała architektura stanowi podstawę do stworzenia narzędzia podpowiadającego możliwe duplikaty dla zgłaszanego formularza. Kolejny etap prac powinien zawierać wykorzystanie pól kategorycznych w procesie oceny próbek oraz filtry czasowe biorące pod uwagę czas wystawienia formularza oraz potencjalnego duplikatu.

Literatura

[1] The cech complex in topological data analysis. https://jdc.math.uwo.ca/TDA/Doherty-Cech-complex.pdf. Data dostępu: 2022-08-20.

[2] A friendly introduction to siamese networks. https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942. Data dostępu: 2022-08-20.

[3] Perplexity. https://en.wikipedia.org/wiki/Perplexity. Data dostępu: 2022-09-01.

[4] Triplet loss. https://deepchecks.com/glossary/triplet-loss/. Data dostępu: 2022-08-20.

[5] Umap: Uniform manifold approximation and projection for dimension reduction. https://umap-learn.readthedocs.io/en/latest/index.html. Data dostępu: 2022-08-20.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc. , Springer Us.

[7] Jason Brownlee. Why one-hot encode data in machine learning? https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/. Data dostępu: 2022-09-01.

[8] Andriy Burkov. *Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[10] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002.

[11] Oliver Knill. Geometry and distance. https://people.math.harvard.edu/~knill//teaching/summer2012/handouts/week1.pdf, 2012.

[12] Hobson Lane, Hannes Hapke, and Cole Howard. *Przetwarzanie języka naturalnego w akcji*. PWN.

[13] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. 2018.

[14] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

[15] Stuart Russell Peter Norvig. *Artificial Intelligence: A Modern Approach, Global Edition*. Pearson Education Limited.

[16] Vahid Mirjalili Sebastian Raschka. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition*. Helion.

[17] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.