

**Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej**

Praca dyplomowa

*Rozpoznawanie tzw. „fake newsów”*

*na wąskim obszarze tematycznym*

*Recognition of so called „fake news”*

*on specified topic scope*

Autor: *Konrad Daniel Golemo*

Kierunek studiów: *Automatyka i Robotyka*

Opiekun pracy: *dr hab. inż. Jerzy Baranowski*

Kraków, *2022*

**Spis treści**

[1. Wstęp 4](#_Toc92477732)

[1.1. Wprowadzenie 4](#_Toc92477733)

[1.2. Cel i zakres pracy 7](#_Toc92477734)

[1.3. Układ pracy 7](#_Toc92477735)

[2. Analiza aktualnego stanu wiedzy 8](#_Toc92477736)

[2.1. Uczenie maszynowe 8](#_Toc92477737)

[2.2. Przetwarzanie języka naturalnego 8](#_Toc92477738)

[2.3. Wyniki realizacji podobnych projektów 10](#_Toc92477739)

[2.3.1. Detekcja tzw. „fake newsów” w języku angielskim 10](#_Toc92477740)

[2.3.2. Detekcja tzw. „fake newsów” w języku niemieckim 12](#_Toc92477741)

[2.3.3. Detekcja tzw. „fake newsów” w języku hiszpańskim 13](#_Toc92477742)

[3. Realizacja projektu 14](#_Toc92477743)

[3.1. Wybór obszaru tematycznego i tworzenie zbiór danych 14](#_Toc92477744)

[3.2. Przetwarzanie zbioru danych 17](#_Toc92477745)

[3.2.1. Oczyszczanie danych 17](#_Toc92477746)

[3.2.2. Ekstrakcja i selekcja cech tekstu 18](#_Toc92477747)

[3.3. Wybrane algorytmy klasyfikacji 20](#_Toc92477748)

[3.3.1. Naiwny klasyfikator bayesowski 20](#_Toc92477749)

[3.3.2. Liniowa maszyna wektorów nośnych 21](#_Toc92477750)

[3.3.3. Las losowy 22](#_Toc92477751)

[3.4. Model klasyfikatora 23](#_Toc92477752)

[3.5. Tworzenie aplikacji webowej 24](#_Toc92477753)

[4. Rezultaty 25](#_Toc92477754)

[4.1. Testy jednostkowe funkcji 25](#_Toc92477755)

[4.2. Ewaluacja algorytmów klasyfikacji 26](#_Toc92477756)

[4.2.1. Zastosowane metryki 26](#_Toc92477757)

[4.2.2. Naiwny klasyfikator bayesowski 27](#_Toc92477758)

[4.2.3. Liniowa maszyna wektorów nośnych 28](#_Toc92477759)

[4.2.4. Las losowy 29](#_Toc92477760)

[4.3. Aplikacja webowa 30](#_Toc92477761)

[5. Podsumowanie 31](#_Toc92477762)

[5.1. Zrealizowane założenia projektowe 31](#_Toc92477763)

[5.2. Wnioski 31](#_Toc92477764)

[5.3. Kierunki dalszego rozwoju 31](#_Toc92477765)

[6. Bibliografia 32](#_Toc92477766)

# Wstęp

## Wprowadzenie

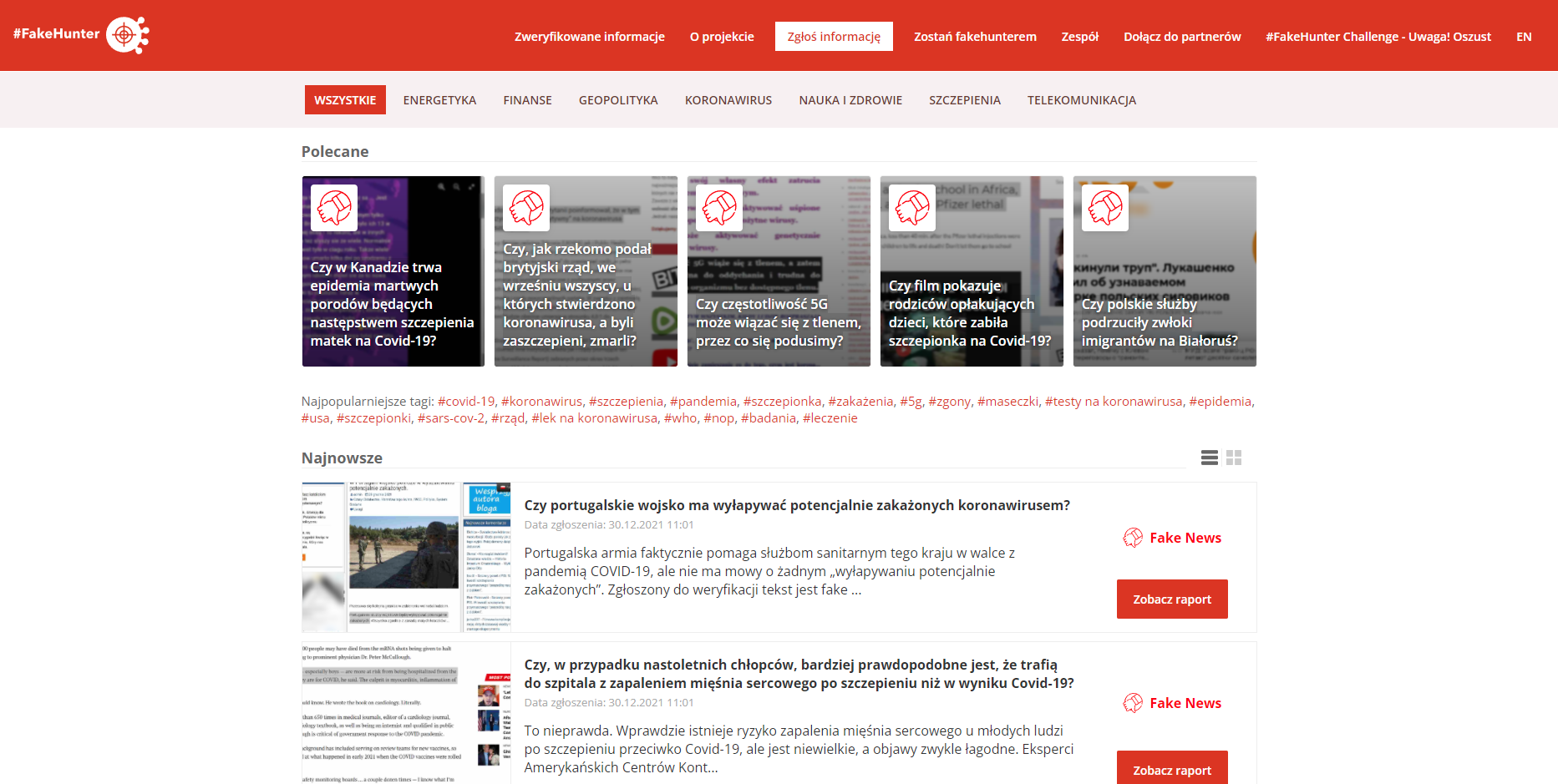
Pojęcie tzw. „fake newsów” odnosi się do fałszywych lub celowo mylących treści publikowanych jako prawdziwe wiadomości. Znaczny wzrost zainteresowania tym zagadnieniem nastąpił w roku 2016, po wyborach prezydenckich w Stanach Zjednoczonych i referendum w Wielkiej Brytanii w sprawie opuszczenia Unii Europejskiej. [1]

W celu odróżnienia informacji prawdziwych od fałszywych (tzw. „fake newsów”) przeprowadzany jest proces tzw. „fact-checkingu”, którego elementy to np. weryfikacja cytatów czy sprawdzanie wiarygodności źródeł. [2]

W Polsce funkcjonuje kilka organizacji weryfikujących informacje, m.in.:

* FakeHunter – serwis tworzony przez Polską Agencję Prasową i GovTech Polska [3],
* FakeNews.pl – portal fundacji „Przeciwdziałamy Dezinformacji” [4],
* AntyFAKE – projekt prowadzony przez Grupę Iberion [5],
* Stowarzyszenie Demagog – pierwsza w Polsce organizacja tego typu [6],
* Konkret24 – serwis należący do Grupy TVN [7].

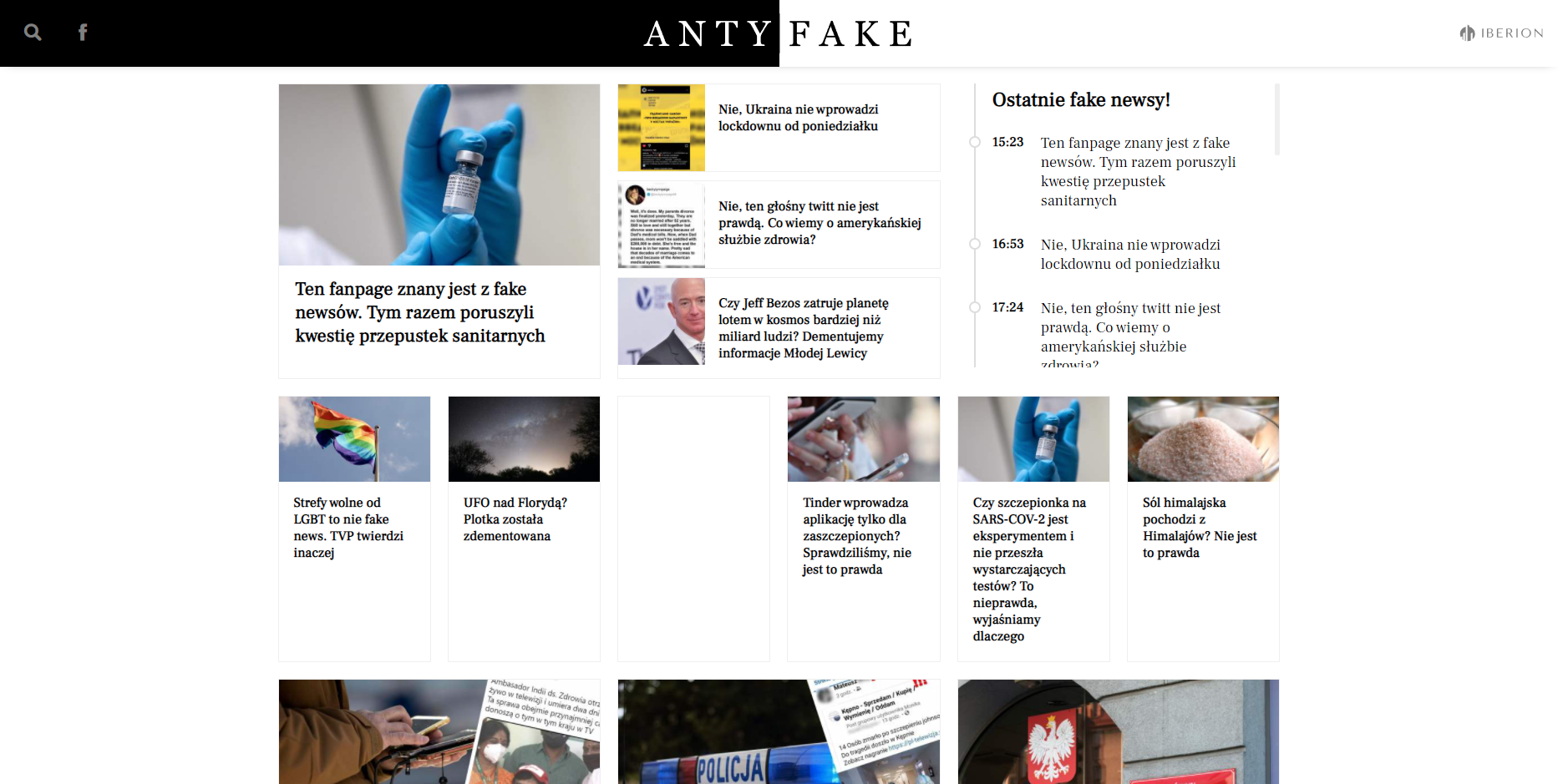
Na Rysunkach 1.1–1.5 przedstawiono strony główne wymienionych portali.



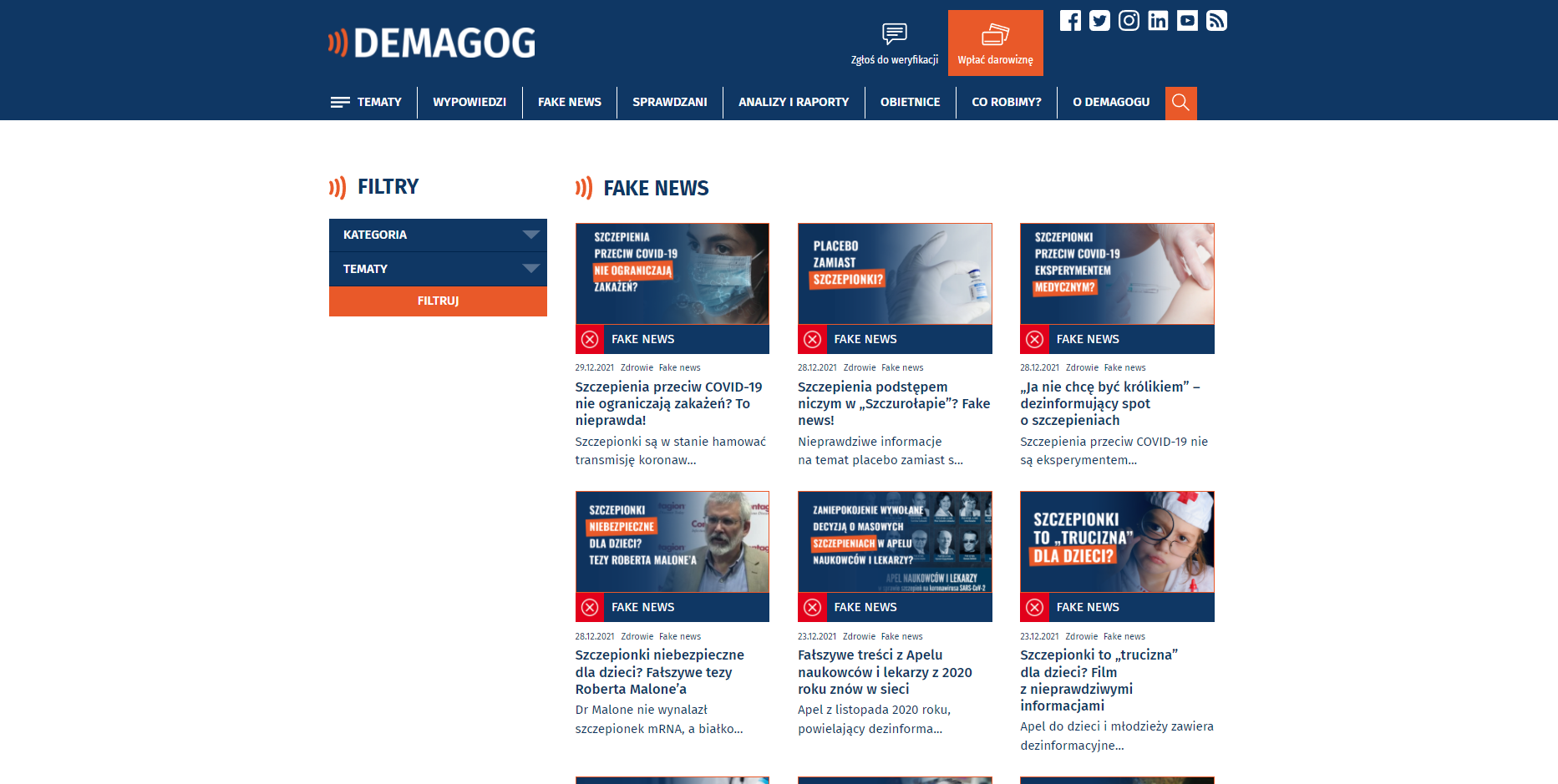
**Rysunek 1.1.** Strona główna portalu FakeHunter [3]



**Rysunek 1.2.** Strona główna portalu FakeNews.pl [4]



**Rysunek 1.3.** Strona główna portalu AntyFAKE [5]



**Rysunek 1.4.** Strona główna portalu Demagog [6]



**Rysunek 1.5.** Strona główna portalu Konkret24 [7]

## Cel i zakres pracy

Celem niniejszej pracy jest zastosowanie podstawowych metod klasyfikacji tekstu do rozpoznawania tzw. „fake newsów” dotyczących pewnego wybranego obszaru tematycznego, poprzez wykonanie następujących kroków:

1. Dokonanie analizy aktualnego stanu wiedzy.
2. Wybór obszaru tematycznego.
3. Stworzenie korpusu tekstów.
4. Wybór i zastosowanie algorytmów uczenia maszynowego.
5. Zaprojektowanie aplikacji i przeprowadzenie testów.

## Układ pracy

Niniejsza praca zorganizowana jest w następujący sposób: Rozdział 1 zawiera wprowadzenie w tematykę, cel i zakres pracy wraz z jej układem. W Rozdziale 2 przeprowadzono analizę aktualnego stanu wiedzy – opisano rozważane zagadnienia i dokonano przeglądu realizacji podobnych projektów. Rozdział 3 dotyczy wykonania projektu – uzasadniono wybór obszaru tematycznego, omówiono proces gromadzenia i przygotowania danych, przedstawiono wybrane algorytmy klasyfikacji a także model kompletnego klasyfikatora. W Rozdziale 4 zaprezentowano rezultaty testów jednostkowych stworzonego oprogramowania oraz wyniki ewaluacji zaprojektowanego klasyfikatora. Rozdział 5 zawiera opis zrealizowanych celów i założeń pracy, wnioski oraz propozycję kierunków możliwego rozwoju.

# Analiza aktualnego stanu wiedzy

## Uczenie maszynowe

Zgodnie z [8] uczenie maszynowe (ang. *Machine Learning, ML*) to ogół metod, za pomocą których na podstawie dostępnych danych dokonywana jest pewna predykcja. Wyróżnia się trzy główne grupy uczenia maszynowego:

* uczenie nadzorowane (ang. s*upervised learning*) – w przypadku zbioru danych wejściowych ze znanymi oczekiwanymi wyjściami (etykietami w klasyfikacji, wartościami liczbowymi w regresji),
* uczenie nienadzorowane (ang. *unsupervised learning*) – w przypadku zbioru danych wejściowych bez określonych oczekiwanych wyjść,
* uczenie przez wzmacnianie (ang. *reinforcement learning*) – w przypadku, gdy dane wejściowe pobierane są ze środowiska, w którym funkcjonuje model, w celu maksymalizacji pewnej nagrody.

Schemat typowego postępowania w realizacji nadzorowanego uczenia maszynowego, składa się z następujących kroków:

1. Gromadzenie danych.
2. Wykorzystanie danych w algorytmie uczenia maszynowego.
3. Zastosowanie stworzonego modelu do wyznaczenia predykcji w oparciu o nowe dane.

Procedura ta, w kontekście niniejszej pracy, została szerzej opisana w Rozdziale 3.

## Przetwarzanie języka naturalnego

Przetwarzanie języka naturalnego (ang. *Natural Language Processing, NLP*) to dziedzina badań zajmująca się wykorzystaniem komputerów w analizie tekstu i mowy ludzkiej, łącząca elementy m.in. informatyki, lingwistyki i sztucznej inteligencji. [9]

Do zastosowań przetwarzania języka naturalnego można zaliczyć [10] [11]:

* klasyfikację tekstu (np. detekcja tzw. „fake newsów”, filtracja spamu),
* tłumaczenie automatyczne,
* pozyskiwanie informacji z tekstu (np. rozpoznawanie nazw),
* podsumowania tekstów, systemy odpowiadania na pytania,
* systemy dialogowe (tzw. „czatboty”).

Zgodnie z [8] [12] w przypadku klasyfikacji tekstu, a więc zadania nadzorowanego uczenia maszynowego, schemat typowego postępowania składa się z następujących kroków:

1. Gromadzenie danych.
2. Wstępne przetwarzanie danych.
3. Ekstrakcja i selekcja cech tekstu.
4. Wykorzystanie algorytmu uczenia maszynowego.

Na etapie wstępnego przetwarzania danych do najczęściej stosowanych metod należą:

* tokenizacja – podział sekwencji znaków na tokeny (poszczególne wyrazy lub zdania),
* filtracja – usunięcie wyrazów należących do listy tzw. „stop-words”, czyli często pojawiających się w tekście, lecz nie przekazujących informacji,
* lematyzacja – sprowadzenie wyrazów do ich podstawowych form (czasowników do formy bezokolicznika, rzeczowników do mianowników liczby pojedynczej),
* tzw. „stemming” – usunięcie formantów fleksyjnych wyrazów.

Ekstrakcja cech tekstu zazwyczaj przeprowadzana jest za pomocą konwersji dokumentów do wektorów w przestrzeni cech, w których z każdym wyrazem (tokenem) związana jest pewna wartość liczbowa (waga tego wyrazu). Wyróżnia się dwa modele doboru wag:

* model boolowski – do każdego wyrazu należącego do dokumentu przypisana jest pewna dodatnia waga, a do każdego nienależącego – wartość 0,
* metoda TF-IDF (ang. *term frequency-inverse document frequency*) – wagi wyznaczane są na podstawie liczby wystąpień wyrazów.

Do stosowanych algorytmów klasyfikacji zalicza się:

* naiwny klasyfikator bayesowski (ang. *Naive Bayes Classifier, NB*) – klasyfikator probabilistyczny, zakładający niezależność występowania wyrazów w tekście, szerzej opisany w Rozdziale 3.3.1,
* liniową maszynę wektorów nośnych (ang. *Linear* *Support Vector Machine, LSVM*) –algorytm separujący dostępne klasy za pomocą pewnej hiperpłaszczyzny, szerzej opisany w Rozdziale 3.3.2,
* drzewo decyzyjne (ang. *Decision Tree, DT*) – bazujące na rekurencyjnym podziale danych, szerzej opisane w Rozdziale 3.3.3,
* algorytm -najbliższych sąsiadów (ang. *K-Nearest Neighbour Classfier, KNN*) – klasa wyznaczana jest w oparciu o podobieństwo dokumentu do jego sąsiedztwa.

Całość wspomnianej procedury, w kontekście niniejszej pracy, została szerzej opisana w Rozdziale 3.

## Wyniki realizacji podobnych projektów

### Detekcja tzw. „fake newsów” w języku angielskim

W publikacji [13] wykorzystano zbiór danych zgromadzony przez serwis BuzzFeed News, zawierający 2282 artykułów o tematyce politycznej przypisanych ręcznie do jednej z czterech kategorii: „głównie prawda”, „głównie fałsz”, „mieszanina prawdy i fałszu” oraz „brak treści merytorycznej”. Podczas implementacji klasyfikatora zredukowano liczebność zestawu poprzez odrzucenie artykułów nienależących do grupy „głównie prawda” lub „głownie fałsz” oraz rekordów niezawierających żadnego tekstu. Finalnie otrzymano zbiór zawierający łącznie 1771 artykułów, spośród których ok. 5% stanowiły tzw. „fake newsy”.

Do wykrywania tekstów zawierających nieprawdziwe informacje wykorzystano naiwny klasyfikator bayesowski (Rozdział 2.2 i 3.3.1). Dane podzielono na trzy zestawy: zbiór uczący, walidacyjny i testowy. W zaproponowanej procedurze klasyfikacji iterowano po każdym wyrazie w analizowanym artykule, a o przydzieleniu do jednej z dwóch grup (prawdziwy albo fałszywy) decydowało prawdopodobieństwo warunkowe wyznaczone na podstawie zawartych w tekście pewnych konkretnych wyrazów. Po wytrenowaniu klasyfikatora i dostrojeniu jego parametrów przystąpiono do testów i ewaluacji działania algorytmów za pomocą dokładności (Rozdział 4.2.1), w wyniku której ogółem uzyskano wynik ok. 75% dokładności klasyfikacji (Tabela 2.1).

W celu poprawy działania klasyfikatora zaproponowano m.in. usunięcie z artykułów wyrazów należących do listy tzw. „stop-words” (Rozdział 2.2), wykorzystanie tzw. „stemmingu” (Rozdział 2.2), czy zmianę sposobu wyznaczania prawdopodobieństwa.

**Tabela 2.1.** Rezultaty ewaluacji klasyfikacji przeprowadzonej na zbiorze testowym [13]

|  |  |
| --- | --- |
| **Rodzaj artykułu** | **Dokładność []** |
| Prawdziwy | 76 |
| Fałszywy | 72 |
| Ogółem | 75 |

W pracy [14] zgromadzono 25 200 artykułów o tematyce politycznej, spośród których połowa stanowią wiadomości prawdziwe, a długość każdego z tekstów jest nie mniejsza niż 200 znaków. Dane podzielono na zbiór treningowy i testowy. Do określenia przestrzeni cech użyto modelu -gramowego, który na podstawie wyrazów przewiduje kolejny [11], a następnie zredukowano liczbę tych cech przy pomocy metody TF (Rozdział 3.2.2) oraz TF-IDF (Rozdział 2.2 i 3.2.2).

Klasyfikację przeprowadzono m.in. za pomocą liniowej maszyny wektorów nośnych (Rozdział 2.2 i 3.3.2), algorytmu -najbliższych sąsiadów (Rozdział 2.2) oraz drzewa decyzyjnego (Rozdział 2.2 i 3.3.3). W trakcie trenowania algorytmów wykorzystano 5-krotny sprawdzian krzyżowy (ang. *k-fold cross-validation*, metoda polegająca na -krotnym losowym podziale oryginalnego zbioru treningowego na mniejszy podzbiór treningowy i testowy, na których uczony jest algorytm uczenia maszynowego [11]).

W eksperymencie porównano dokładność klasyfikacji (Rozdział 4.2.1) dla obu metod ekstrakcji cech przy zmiennym rozmiarze -gramu dla każdego z zastosowanych algorytmów. Największą dokładność na poziomie 92% osiągnięto dla algorytmu LSVM (Tabela 2.2).

**Tabela 2.2.** Rezultaty ewaluacji klasyfikacji przeprowadzonej na zbiorze testowym   
z wykorzystaniem algorytmu LSVM [14]

|  |  |  |  |
| --- | --- | --- | --- |
| **Rozmiar**  **-gramu** | **Liczba**  **cech** | **Dokładność []** | |
| **TF-IDF** | **TF** |
| Unigram | 1000 | 89 | 87 |
| 5000 | 89 | 87 |
| 10000 | 89 | 87 |
| 50000 | **92** | 87 |
| Bigram | 1000 | 87 | 86 |
| 5000 | 87 | 83 |
| 10000 | 88 | 82 |
| 50000 | **89** | 82 |

W artykule [15] wykorzystano ten sam zbiór danych, co w pozycji [13] (2282 tekstów o tematyce politycznej), lecz zaproponowano inną metodę podziału na treści wiarygodne i nieprawdziwe. Odrzucono wiadomości oznaczone etykietą „brak treści merytorycznej”, a te należące do grup „głównie fałsz” i „mieszanina prawdy i fałszu” połączono i potraktowano jako teksty fałszywe (342 pozycje). Pozostałe, należące do klasy „głównie prawda”, przydzielono do kategorii artykułów prawdziwych (1666 pozycje).

Klasyfikacji dokonano poprzez zastosowanie m.in: naiwnego klasyfikatora bayesowskiego (Rozdział 2.2 i 3.3.1), lasu losowego (ang. *Random Forest, RF*) (Rozdział 3.3.3), oraz XGBoost (metoda wzmacniająca gradient, bazująca na drzewach [16]). Cechy artykułów wyznaczono ręcznie, wyróżniając trzy kategorie: cechy tekstu (m.in. właściwości językowe), cechy źródła danych (m.in. uprzedzenia polityczne czy wiarygodność) i cechy otoczenia (liczba interakcji użytkowników z postem).

Ewaluację klasyfikacji przeprowadzono za pomocą współczynnika -score (Rozdział 4.2.1) oraz obliczonego pola pod krzywą ROC (ang. *Receiver Operating Characteristic*) (Rozdział 4.2.1). Najlepsze rezultaty uzyskano dla algorytmu RF i XGBoost (Tabela 2.3).

**Tabela 2.3.** Rezultaty ewaluacji klasyfikacji przeprowadzonej na zbiorze testowym   
przy pomocy pola pod krzywą ROC oraz współczynnika -score [15]

|  |  |  |
| --- | --- | --- |
| **Algorytm  klasyfikacji** | **Pole pod  krzywą ROC []** | **Współczynnik  -score []** |
| NB | 72 | 75 |
| RF | **85** | **81** |
| XGB | **86** | **81** |

### Detekcja tzw. „fake newsów” w języku niemieckim

W artykule [17] zgromadzono zbiór danych zawierający 490 artykułów przekazujących nieprawdziwe informacje oraz 4500 wiadomości opublikowanych przez wiarygodnych wydawców głównego nurtu. Zastosowano dwie metody klasyfikacji: jedną opierającą się na maszynie wektorów nośnych (Rozdział 2.2 i 3.3.2), drugą na konwolucyjnej sieci neuronowej (ang. *Convolutional Neural Network, CNN*) (wielowarstwowa sieć neuronowa wykorzystująca operacje konwolucji, często stosowana w przypadku dużej ilości danych [18]). Do wytrenowania klasyfikatorów wykorzystano teksty zawierające co najmniej 200 słów, co zredukowało liczbę tzw. „fake newsów” do 300.

W przypadku klasyfikacji z wykorzystaniem maszyny wektorów nośnych połowę zbioru artykułów wykorzystano do treningu, drugą połowę przeznaczając na testy. Na podstawie zestawu danych, za pomocą metody TF-IDF (Rozdział 2.2 i 3.2.2) określono wagę słów w tekście i wyznaczono odpowiadające im wektory cech. W ramach klasyfikatora opierającego się na CNN dane podzielono na trzy zestawy – treningowy (50% zbioru), walidacyjny (25%) i testowy (25%), a w celu wytrenowania architektury sieci neuronowej wykorzystano wcześniej przygotowane tzw. „word embeddings” – pewną wektorową reprezentację słów.

W wyniku ewaluacji za pomocą współczynnika -score (Rozdział 4.2.1), dla klasyfikatora z algorytmem SVM otrzymano , a dla metody wykorzystującej CNN (Tabela 2.4).

**Tabela 2.4.** Rezultaty ewaluacji klasyfikacji przeprowadzonej na zbiorze testowym   
przy pomocy dokładności oraz współczynnika -score [17]

|  |  |  |
| --- | --- | --- |
| **Algorytm  klasyfikacji** | **Pole pod  krzywą ROC** | **Współczynnik  -score []** |
| SVM | 0,96 | 74 |
| CNN | **0,98** | **89** |

### Detekcja tzw. „fake newsów” w języku hiszpańskim

W pracy [19] zebrano łącznie 971 artykułów z 9 kategorii tematycznych, w tym 491 oznaczonych jako prawdziwe i 480 jako fałszywe, które następnie podzielono na zestawy treningowe (70% zbioru) i testowe (30%). Jako reprezentację cech wykorzystano m.in.:

* tzw. „bag-of-words” (BOW) – metoda reprezentowania dokumentów za pomocą wektora, w którym ciąg 0 i 1 określa występowanie wyrazów z pewnego słownika [11],
* tzw. „part-of-speech tagging” (POS) – metoda przypisywania do każdego wyrazu odpowiadającej mu części mowy [11],
* znakowe -gramy – metoda podziału każdego wyrazu na -znakowe części [20],

a do klasyfikacji wykorzystano algorytmy: maszynę wektorów nośnych (Rozdział 2.2 i 3.3.2), regresję logistyczną (ang. *Logistic Regression, LR*) (metoda regresji umożliwiająca wyznaczenie prawdopodobieństwa należenia do klasy [11]), las losowy (Rozdział 3.3.3) oraz tzw. „boosting” (metoda polegająca na stworzeniu dobrego algorytmu na podstawie słabych [21]).

Korzystając z dokładności (Rozdział 4.2.1) porównano działanie klasyfikatorów trenowanych na różnych zbiorach cech tekstu – BOW, POS oraz połącznie obu (Tabela 2.5).

**Tabela 2.5.** Rezultaty ewaluacji klasyfikacji przeprowadzonej na zbiorze testowym [19]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reprezentacja cech** | **Dokładność klasyfikacji []** | | | |
| **SVM** | **LR** | **RF** | **BO** |
| BOW | 72 | 72 | 76 | 73 |
| POS | 68 | 67 | 64 | 61 |
| BOW+POS | 71 | 74 | **77** | 72 |

Zestawiono także dokładność klasyfikacji modeli bazujących na znakowych -gramach z usuwaniem wyrazów należących do listy tzw. „stop-words” i nie (Tabela 2.6).

**Tabela 2.6.** Rezultaty ewaluacji klasyfikacji przeprowadzonej na zbiorze testowym   
z wykorzystaniem modelów -gramowych [19]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rozmiar**  **znakowego**  **-gramu** | **Dokładność klasyfikacji []** | | | | | | | |
| **usuwanie wyrazów** | | | | **nieusuwanie wyrazów** | | | |
| **SVM** | **LR** | **RF** | **BO** | **SVM** | **LR** | **RF** | **BO** |
| -gram | 69 | 67 | 70 | 70 | 73 | 71 | 75 | 75 |
| -gram | 72 | 70 | 71 | 72 | 76 | 77 | 75 | **77** |
| -gram | 74 | 70 | 74 | 73 | 76 | 76 | 76 | 76 |

W publikacji lepsze rezultaty uzyskano w przypadku korzystania z reprezentacji -gramowej – 77% dokładności dla znakowego -gramu o rozmiarze 4 i metody „boosting” i 77% dokładności dla kombinacji BOW z POS i algorytmu RF).

# Realizacja projektu

## Wybór obszaru tematycznego i tworzenie zbiór danych

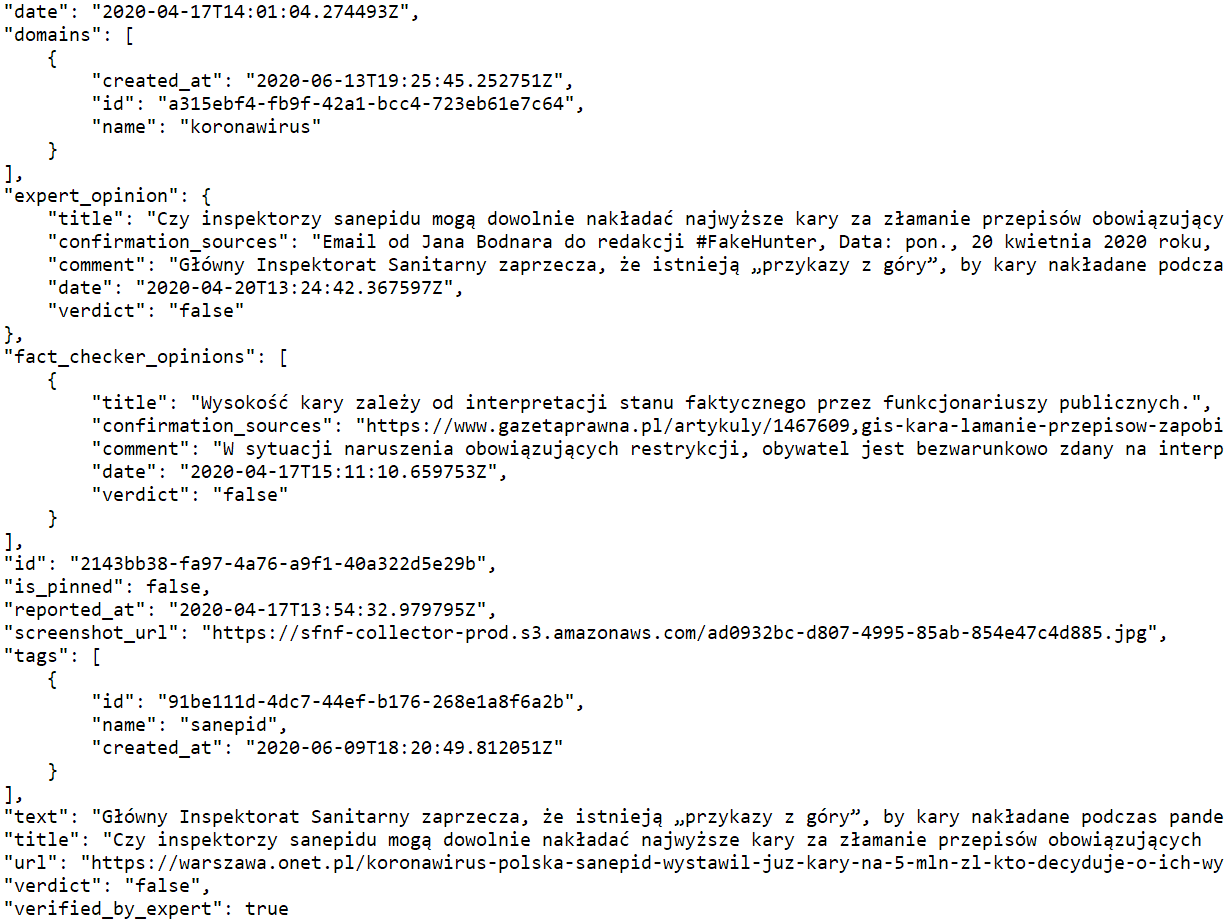
Na początku roku 2020 w związku z pandemią COVID-19 zaczęto obserwować znaczną liczbę tzw. „fake newsów” podważających istnienie wirusa czy jego śmiertelność [22]. Z uwagi na mnogość wiadomości dotyczących koronawirusa, zarówno prawdziwych, jak i fałszywych, w niniejszej pracy zdecydowano się skupić właśnie na tym obszarze tematycznym. Do stworzenia korpusu mogącego posłużyć jako dane w procesie klasyfikacji, konieczne było zgromadzenie odpowiedniej liczby tekstów zawierających prawdziwe albo fałszywe informacje. W tym celu wykorzystano tzw. „web scraping”, czyli technikę pozyskiwania informacji z witryn internetowych [23], używając bibliotek Pythona: Requests [24] oraz Beautiful Soup [25].

Nieprawdziwe wiadomości pobrano z portalu FakeHunter [3], na którego łamach, na podstawie wiarygodnych źródeł, weryfikowana jest poprawność informacji przesłanych przez użytkowników. Przeanalizowane doniesienia są następnie publikowane w formie artykułów zawierających werdykt („Fake News” albo „Prawda”) oraz raport eksperta wraz ze zdjęciem całości zgłoszonego tekstu lub jego fragmentu, a także link do jego źródła. Przykładowa analiza opublikowana w serwisie została przestawiona na Rysunku 3.1.



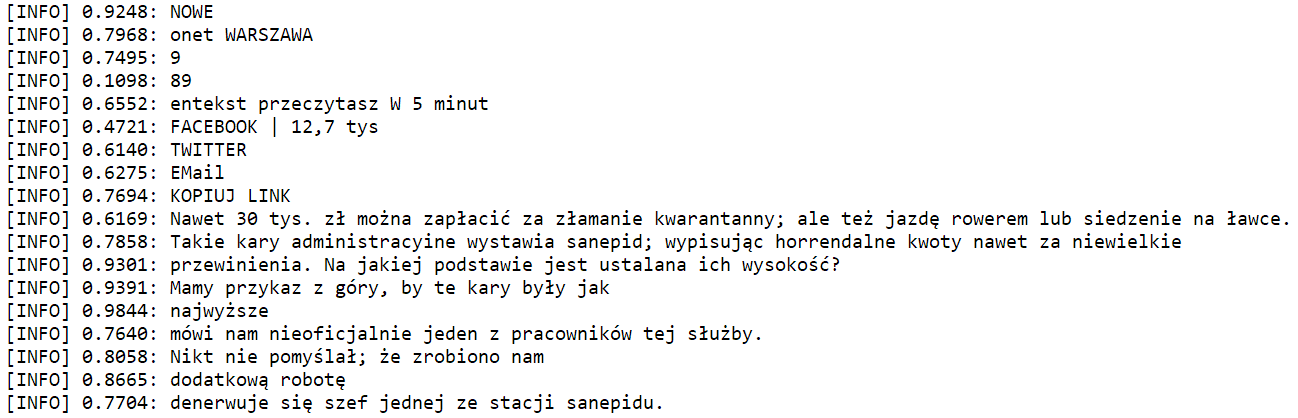
**Rysunek 3.1.** Raport z weryfikacji opublikowany na łamach portalu FakeHunter [26]

W celu pobrania weryfikacji wykorzystano tzw. „API” (ang. *application programming interface*) portalu, umożliwiające akwizycję zawartości strony do postaci słownika składającego się z par klucz-wartość (np. do klucza „werdykt” przypisana jest wartość „fałszywy”). W ten sposób uzyskano dostęp do elementów strony widocznych na Rysunku 3.2.



**Rysunek 3.2.** Raport z weryfikacji [26] widoczny w tzw. „API”, z widocznym polem „opinia eksperta”, zawierającym m.in. werdykt (niebieska ramka), a także elementami zawierającymi adres, pod którym znajduje się zdjęcie weryfikowanego artykułu (zielona ramka) oraz adres źródłowy tego artykułu (pomarańczowa ramka)

Mimo dostępu do witryn, z których pochodzą badane artykuły, niemożliwe było zaprojektowanie mechanizmu dokonującego akwizycji ich treści – każda strona projektowana jest w inny sposób, tekst może znajdować się w różnych jej elementach. Do wyodrębnienia artykułów stanowiących tzw. „fake newsy” użyto zamieszczonych w analizie zdjęć, zawierających część lub całość ich treści. W tym celu wykorzystano oprogramowania rozpoznającego znaki, tzw. „OCR” (ang. *Optical Character Recognition*), z biblioteki EasyOCR [27], którego przykład działania dla domyślnych parametrów i dekodera typu „beam-search” (algorytm przeszukiwania grafu, w którym w każdej iteracji pozostawiana jest jedynie grupa najbardziej obiecujących ścieżek [11]), widoczny jest na Rysunku 3.2.

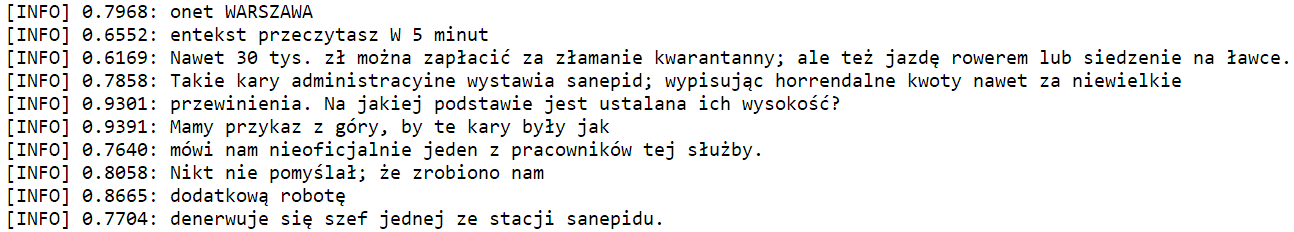


**Rysunek 3.3.** Rezultat ekstrakcji tekstu za pomocą oprogramowania rozpoznającego znaki z artykułu [26] dla jego domyślnych parametrów

Elementy zwracane przez to oprogramowanie to m.in. poziom pewności detekcji obiektu oraz jego treść. W uzyskanym rezultacie problem stanowiło jednak traktowanie jako tekst elementów, które w rzeczywistości nim nie były (np. elementy znajdujące się w prawym górnym rogu Rysunku 3.3), czy detekcja fragmentów odnośników do innych portali lub przycisków.

W celu eliminacji tego typu zakłóceń określono minimalny rozmiar prostokąta otaczającego tekst poprzez ustalenie wartość parametru na 200 (domyślnie równego 10), w wyniku otrzymując detekcje przedstawione na Rysunku 3.4

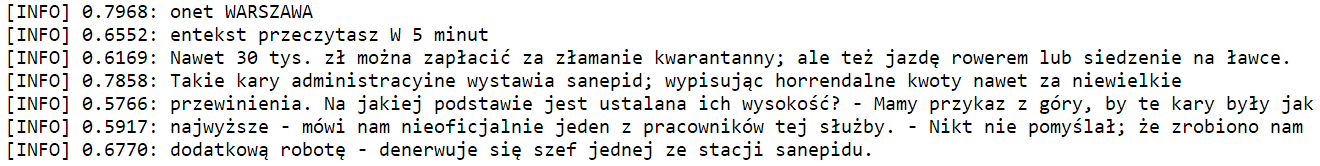




**Rysunek 3.4.** Rezultat ekstrakcji tekstu za pomocą oprogramowania rozpoznającego znaki z artykułu [26] dla parametru

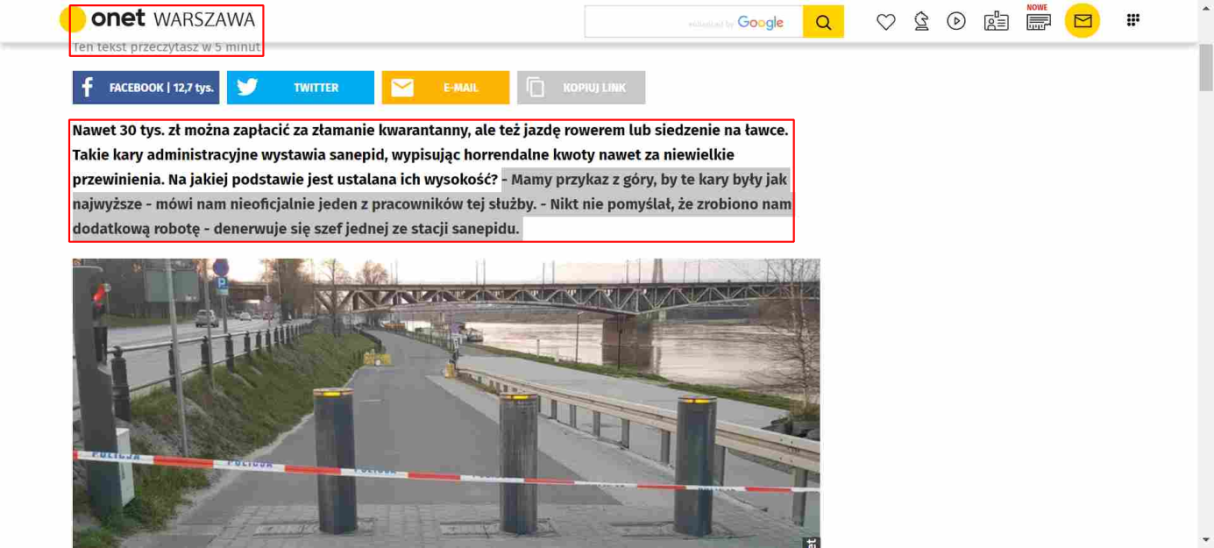
Kolejną przeszkodę stanowiło traktowanie wyrazów rozdzielonych myślnikiem jako osobne obiekty. Za łączenie prostokątów otaczających teksty odpowiada parametr , określający minimalną poziomą odległość między łączonymi prostokątami, którego wartość ustalono na 1 (domyślnie przyjmował 0,5). Rezultat przedstawiono na Rysunku 3.5.

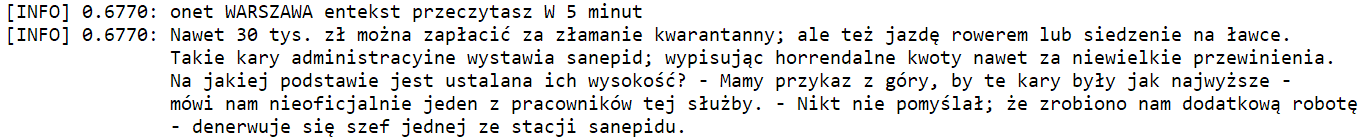




**Rysunek 3.5.** Rezultat ekstrakcji tekstu za pomocą oprogramowania rozpoznającego znaki z artykułu [26] dla parametru i

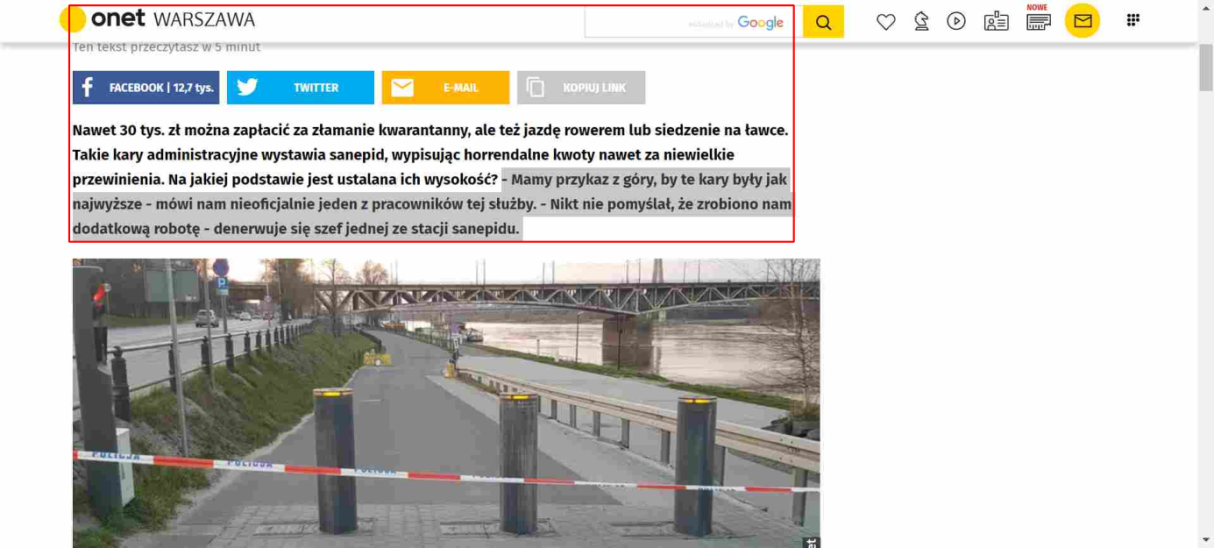
W następnym kroku połączono poszczególne elementy w większe obiekty poprzez ustawienie flagi , w wyniku czego uzyskano detekcję ukazaną na Rysunku 3.6.

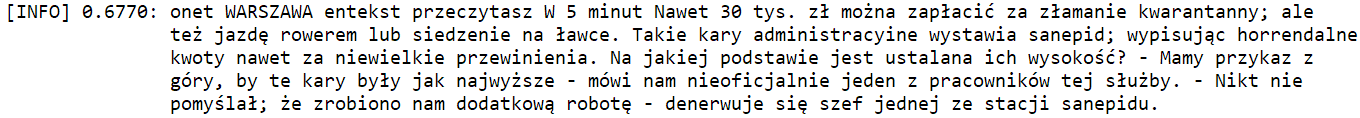




**Rysunek 3.6.** Rezultat ekstrakcji tekstu za pomocą oprogramowania rozpoznającego znaki z artykułu [26] dla flagi , parametru i

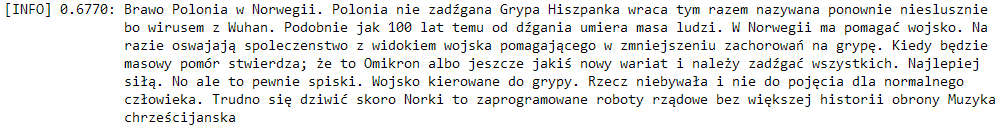
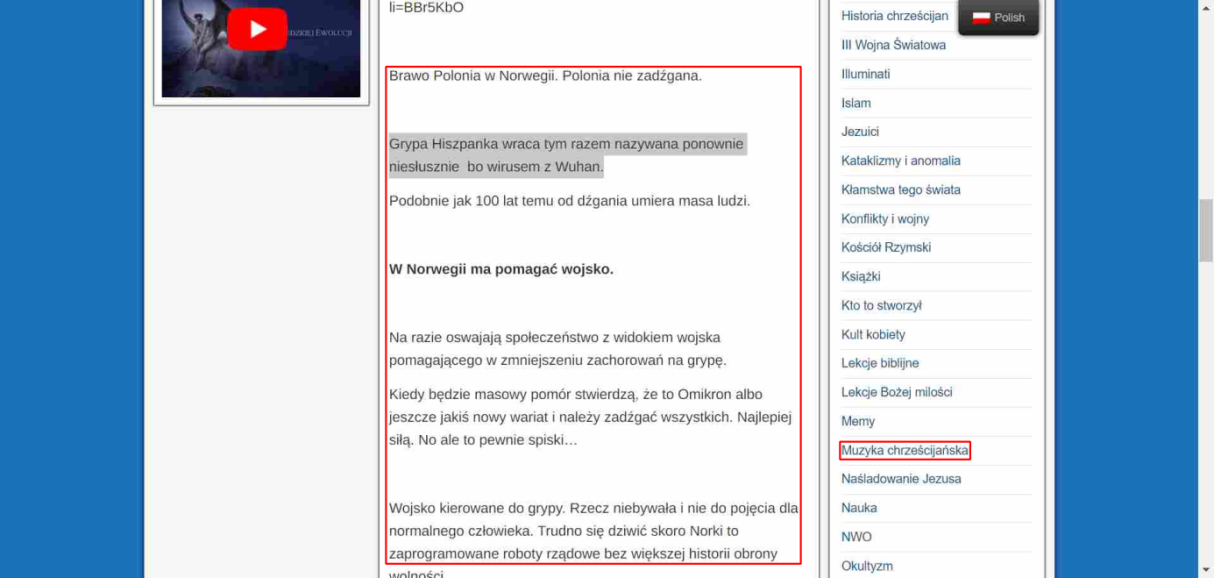
Uznano, że większe obiekty także powinny zostać połączone – w tym przykładzie osobny element stanowi nazwa serwisu, z którego pochodzi artykuł, co nie jest informacją bezużyteczną, a w niektórych przypadkach treść wiadomości może być podzielona większymi przerwami na akapity, które powinny zostać połączone. W tym celu ustalono wartość parametru , określającego maksymalną pionową odległość między łączonymi obiektami, na 4 (domyślnie wartość parametru wynosiła 0,5), uzyskując rezultat widoczny na Rysunku 3.7.



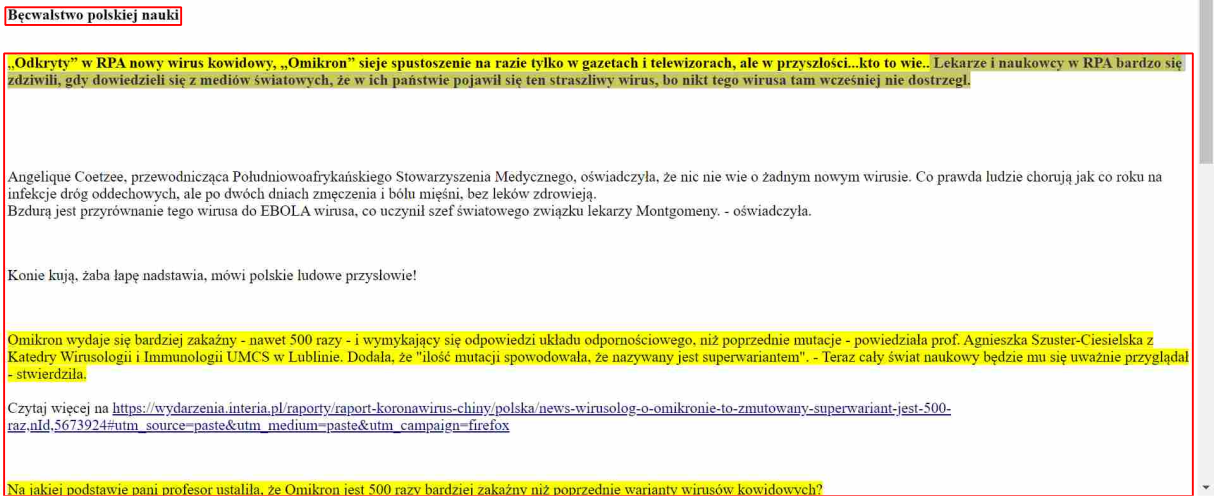


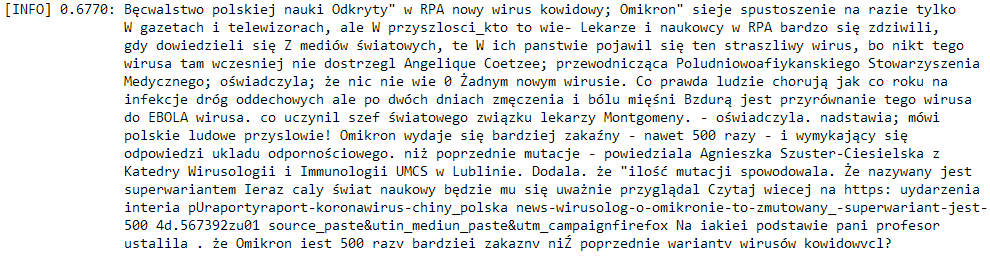
**Rysunek 3.7.** Rezultat ekstrakcji tekstu za pomocą oprogramowania rozpoznającego znaki z artykułu [26] dla flagi , parametru , i

Poniżej, na Rysunku 3.8 i 3.9, zamieszczono przykłady ekstrakcji cech z innych artykułów, na których widoczne są połączone w większe elementy poszczególne akapity tekstu. Na Rysunku 3.8 widoczny jest także element nienależący do treści wiadomości, który będąc częścią detekcji stanowi szum.



**Rysunek 3.8.** Rezultat ekstrakcji tekstu za pomocą oprogramowania rozpoznającego znaki z artykułu [28]





**Rysunek 3.9.** Rezultat ekstrakcji tekstu za pomocą oprogramowania rozpoznającego znaki z artykułu [29]

Pseudokod procedury pobierania danych z portalu FakeHunter przedstawiono poniżej:

**for** strona **in** zakres\_stron:

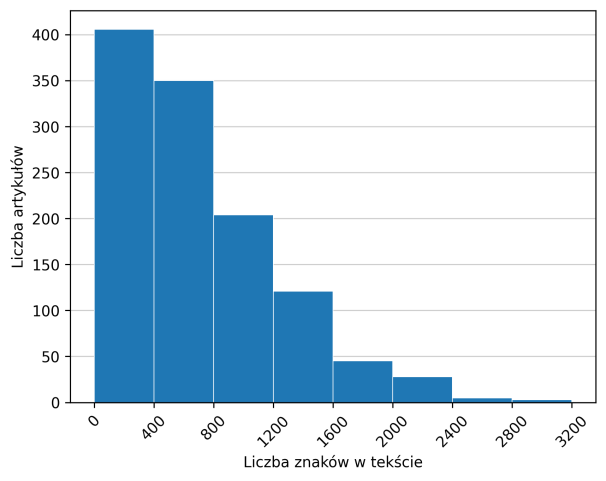
weryfikacje = pobierz\_weryfikacje(strona)

**for** w **in** weryfikacje:

tekst = pobierz\_tekst\_z\_obrazu(weryfikacja[adres\_obrazu])

zapisz\_do\_tabeli(w[werdykt],w[tytuł],tekst,w[adres\_artykułu])

Z portalu wyodrębniono 939 tekstów określonych jako tzw. „fake news”, 151 prawdziwych i 74 bez werdyktu. Artykuły te umieszczono w pliku *fakehunter\_dataset.xlsx*, stanowiącym załącznik do pracy*.* Na Rysunek 3.10 przedstawiono rozkład ich długości.



**Rysunek 3.10.** Histogram długości artykułów pobranych z portalu FakeHunter

Jako bazę wiarygodnych wiadomości przyjęto serwis informacyjny wydawnictwa Termedia [30], publikujący artykuły o tematyce medycznej, w tym dotyczące wirusa SARS-CoV-2. Z uwagi na brak możliwości skorzystania z tzw. „API” serwisu, zawartość została wyodrębniona ze strony z wykorzystaniem parsowania kodu HTML witryny internetowej.

Pseudokod procedury pobierania danych z portalu Termedia przedstawiono poniżej:

**for** strona **in** zakres\_stron:

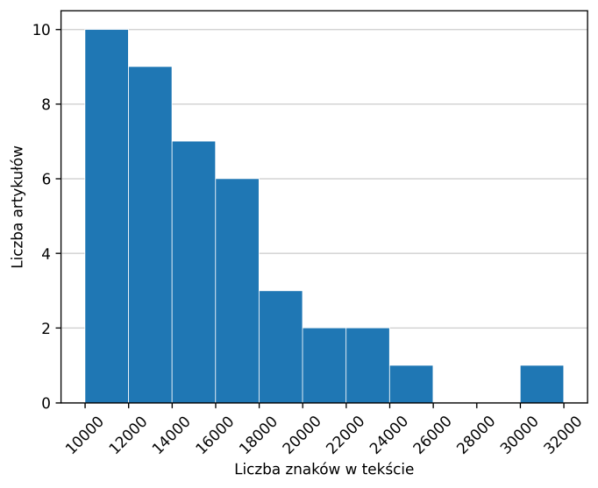
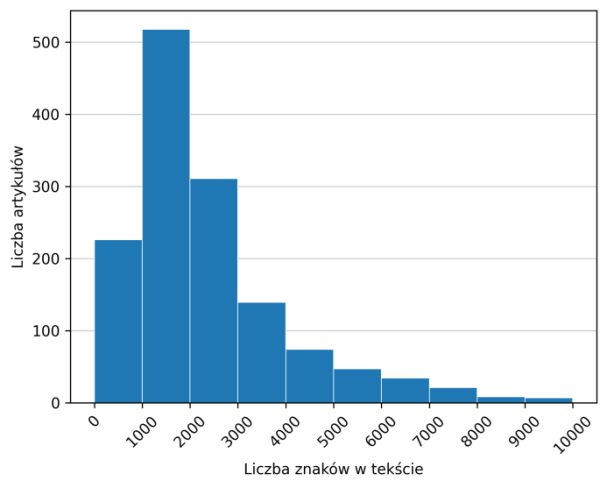
adresy\_artykulów = pobierz\_adresy(strona)

**for** adres **in** adresy\_artykułów:

tytuł, tekst, adres\_artykułu = pobierz\_tekst\_z\_artykulu(adres)

zapisz\_do\_tabeli(‘prawda’, tytuł, tekst, adres\_artykułu)

Z portalu wyodrębniono 1429 artykułów, które umieszczono w pliku *termedia\_dataset.xlsx*, stanowiącym załącznik do pracy*.* Na Rysunek 3.11 przedstawiono długości teksów.



**Rysunek 3.11.** Histogram długości artykułów pobranych z portalu Termedia w zakresie od 0 do 10 tysięcy (po lewej) i od 10 do 32 tysięcy (po prawej)

## Przetwarzanie zbioru danych

### Oczyszczanie danych

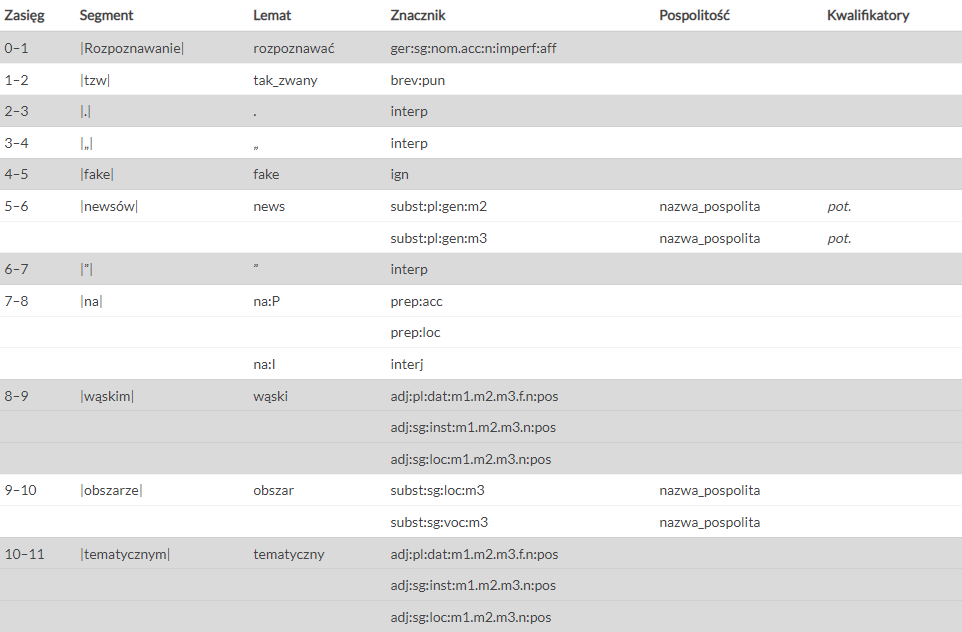
W ramach przygotowania zbioru danych z portalu FakeHunter do konwersji do przestrzeni cech tekstu, konieczne było wykonanie poniższych operacji:

1. Usunięcie rekordów niezawierających treści artykułu – przypadki, w których oprogramowanie wykrywające znaki nie odnalazło tekstu na zdjęciu.
2. Usunięcie tekstów niepolskojęzycznych – przypadki, w których artykuł pochodził z zagranicznego źródła, do ich wykrycia wykorzystano bibliotekę polyglot [31].
3. Usunięcie tekstów bez werdyktu – przypadki, w których zgłoszona wiadomość przekazywała informacje nieweryfikowalne.
4. Usunięcie tekstów błędnie wyodrębnionych z serwisu Twitter – przypadki, w których oprogramowanie wykrywającego znaki niepoprawnie wyodrębniło niektóre teksty pochodzące z serwisu Twitter.

W przypadku artykułów z portalu Termedia, należało jedynie usunąć puste rekordy (przypadki, w których treść artykułu była niedostępna).

Następnie na obu wstępnie przetworzonych zestawach przeprowadzono kolejne operacje:

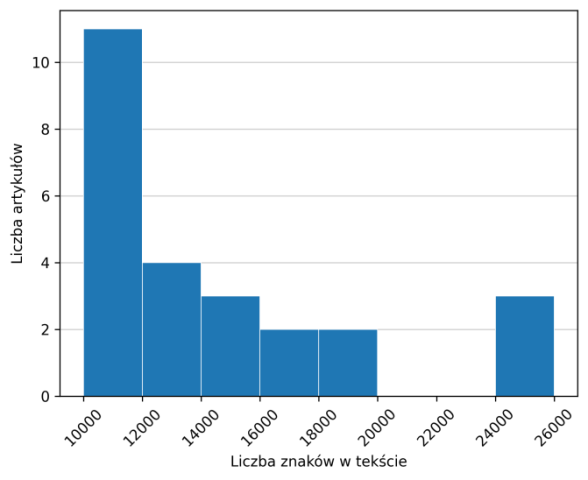
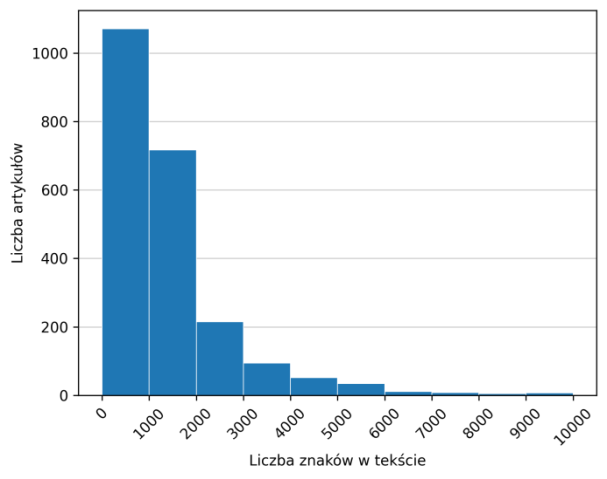
1. Usunięcie znaków ucieczki – znaki tabulacji, nowej linii itp. stanowią jedynie utrudnienie analizy tekstu.
2. Usunięcie znaków interpunkcyjnych i nienależących do polskiego alfabetu – w przetwarzaniu języka naturalnego stanowią one szum, nieprzekazujący żadnej informacji.
3. Zastąpienie wielokrotnych spacji pojedynczą – tokenizacja przeprowadzana jest na podstawie pojedynczych spacji rozdzielających wyrazy, wielokrotne stanowiłby wiec utrudnienie.
4. Zamiana wielkich liter w wyrazach na małe – wielkość liter nie ma znaczenia w przeprowadzanej analizie tekstów.
5. Podział tekstów na tokeny – każdy tekst rozdzielany jest na pojedyncze wyrazy.
6. Usunięcie wyrazów należących do listy tzw. „stop-words” – wyrazy nieprzekazujące informacji często występujące w języku polskim lub będące spójnikami.
7. Przeprowadzenie lematyzacji – w tym celu wykorzystano analizator i generator fleksyjny dla języka polskiego Morfeusz2 [32], którego przykład działania (z pominięciem ww. kroków przygotowawczych) przedstawiono na Rysunku 3.4.



**Rysunek 3.12.** Demonstracja analizy morfologicznej programu Morfeusz2 [32] przeprowadzonej   
dla zdania „Rozpoznawanie tzw. ≫fake newsów≪ na wąskim obszarze tematycznym”

1. Usunięcie rekordów niezawierających treści artykułu – w rezultacie ww. kroków pojawiły się do tej pory niewykryte rekordy bez użytecznego tekstu.
2. Usunięcie tekstów zbyt krótkich – w celu zmniejszenia szumu, zdecydowano się na analizowanie jedynie tekstów zawierających 30 lub więcej znaków.

Finalnie uzyskano zbiór składający się z 1497 artykułów przekazujących prawdziwe informacje i 753 fałszywych wiadomości – łącznie 2240 pozycji. Dane umieszczono w pliku *complete\_dataset.xlsx*, stanowiącym załącznik do pracy. Na Rysunku 3.13 przedstawiono rozkład długości pobranych tekstów, na którym widoczny jest wzrost liczby artykułów o liczbie znaków nieprzekraczającej 1000 związany z dołączeniem do zbioru fałszywych wiadomości.



**Rysunek 3.13.** Histogram długości artykułów w kompletnym zbiorze danych zakresie od 0 do 10 tysięcy (po lewej) i od 10 do 32 tysięcy (po prawej)

### Ekstrakcja i selekcja cech tekstu

W celu konwersji zgromadzonych dokumentów do wektorów cech wykorzystano metodę TF-IDF (ang. *term frequency-inverse document frequency*), która zgodnie z [11] przypisuje tokenom wagi według następującej zależności:

|  |  |  |
| --- | --- | --- |
|  |  | (3.1.1) |

gdzie to częstość występowania wyrazu w dokumencie określona jako stosunek liczby wystąpień wyrazu w dokumencie do liczby wszystkich wyrazów w dokumencie :

|  |  |  |
| --- | --- | --- |
|  |  | (3.1.2) |

a to logarytm dziesiętny ze stosunku liczby wszystkich dokumentów do liczby dokumentów, w których występuje wyraz :

|  |  |  |
| --- | --- | --- |
|  |  | (3.1.3) |

W realizacji projektu wykorzystano metodę TF-IDF z biblioteki scikit-learn, w której składnik IDF wyznaczany jest w następujący sposób [33]:

|  |  |  |
| --- | --- | --- |
|  |  | (3.1.4) |

Wyznaczone wektory TF-IDF są następnie normalizowane z wykorzystaniem normy euklidesowej [33]. Funkcja tworząca wektory cech umożliwia także wyznaczanie ich w oparciu o wyrazowe -gramy, w analizie wykorzystano więc uni- i bigramy.

Zestaw danych został podzielony na zbiór treningowy (70% danych, 1568 tekstów) i testowy (30% danych, 672 teksty), a następnie dokonano wektoryzacji, w wyniku której uzyskano 27416 cech w przypadku unigramów i 193552 dla reprezentacji bigramowej. Selekcję cech przeprowadzono w oparciu o wybór jedynie najczęściej występujących cech pod względem współczynnika w korpusie. W kolejnym kroku zastosowano wymienione w Rozdziale 3.3 algorytmy klasyfikacji.

## Wybrane algorytmy klasyfikacji

### Naiwny klasyfikator bayesowski

Naiwny klasyfikator bayesowski (ang. *Naive Bayes Classifier*) to probabilistyczna metoda klasyfikacji, której zadaniem jest wyznacznie zgodnie z twierdzeniem Bayesa najbardziej prawdopodobnej klasy dla danego wektora zmiennych. Istotnym uproszczeniem znacznie usprawniającym proces uczenia (zwłaszcza przy dużej liczbie zmiennych) jest założenie o wzajemnej niezależność predyktorów, umożliwiające wyznaczanie parametrów dla każdej ze zmiennych oddzielnie. [34]

Zgodnie z [35] prawdopodobieństwo przynależności wektora cech tekstu do klasy jest równe:

|  |  |  |
| --- | --- | --- |
|  |  | (3.2.1) |

Uwzględniając założenie o niezależności predyktorów:

|  |  |  |
| --- | --- | --- |
|  |  | (3.2.2) |

Zgodnie z [36], z uwagi na fakt, że mianownik jest stały, reguła klasyfikacji określona jest zależnością:

|  |  |  |
| --- | --- | --- |
|  |  | (3.2.3) |

a więc najbardziej odpowiednią klasę dla danego wektora cech tekstu stanowi:

|  |  |  |
| --- | --- | --- |
|  |  | (3.2.4) |

gdzie wartości częstość występowania klasy w zbiorze danych () oraz częstość występowania cechy w tekstach należących do klasy () są szacowane na podstawie zestawu danych uczących.

W naiwnym klasyfikatorze bayesowskim z biblioteki scikit-learn, wykorzystanym w realizacji projektu, wartość jest estymowana z uwzględnieniem wygładzania addytywnego, co ma na celu wyeliminowanie zerowania się tego składnika w przypadkach, gdy dana para cecha-klasa nie wystąpiła w zbiorze treningowym. [37]

### Liniowa maszyna wektorów nośnych

Liniowa maszyna wektorów nośnych (ang. *Linear* *Support Vector Machine, LSVM*) to algorytm stosowany w problemach klasyfikacji binarnej, którego celem jest określenie hiperpłaszczyzny rozdzielającej zestaw danych pomiędzy dwie klasy. Wymiar tej hiperpłaszczyzny zależy od liczebności zbioru cech, na podstawie którego będzie ona wyznaczana. Margines wokół granicy (funkcji) decyzyjnej maksymalizowany jest na podstawie par wektorów cech i odpowiadających im etykiet . [38].



**Rysunek 3.14.** Przykład funkcji decyzyjnej dla danych liniowo separowalnych z oznaczonymi ciemniejszym odcieniem wektorami nośnymi; opracowanie własne na podstawie [39]

Zgodnie z [39] zestaw danych uczących , jest liniowo separowalny, jeśli nierówność:

|  |  |  |
| --- | --- | --- |
|  | , | (3.3.1) |

jest spełniona dla wszystkich jego elementów.

Wektory, dla których to tzw. wektory nośne, na podstawie których w przestrzeni cech konstruowana jest optymalna hiperpłaszczyzna określona zależnością:

|  |  |  |
| --- | --- | --- |
|  | , | (3.3.2) |

rozdzielająca dane uczące z największym możliwym marginesem, opisanym wzorem:

|  |  |  |
| --- | --- | --- |
|  | . | (3.3.3) |

Wyznaczanie hiperpłaszczyzny (2.2) sprowadza się do rozwiązania problemu programowania kwadratowego – minimalizacji iloczynu przy ograniczeniach (2.1).

Maszyna wektorów nośnych z biblioteki scikit-learn, wykorzystana w realizacji projektu, przy obliczaniu współczynników hiperpłaszczyzny uwzględnia dodatkowo funkcję straty typu hinge loss, kontrolującą wpływ predykcji ze zbyt małym marginesem lub błędnych. [40]

### Las losowy

Las losowy (ang. *Random Forest, RF*) to metoda uczenia maszynowego działająca w oparciu o grupę drzew decyzyjnych (ang. *Decision Tree*).

Zgodnie z [41], algorytm drzewa decyzyjnego dokonuje rekurencyjnego, binarnego podziału przestrzeni cech według poszczególnych zmiennych objaśniających (predyktorów), aż do momentu spełnienia kryterium zatrzymania. W rezultacie otrzymywane są niepodzielone węzły, zwane końcowymi, na podstawie których określana jest predykcja algorytmu. Do zalet drzew decyzyjnych należą m.in.:

* możliwość stosowania w problemach klasyfikacji i regresji,
* możliwość opisania zależności między zmiennymi objaśniającymi,
* odporność na braki danych, odstające wartości i predyktory bez znaczenia.

Istotną wadę drzew decyzyjnych stanowi m.in. niestabilność (nieznaczna zmiana w przestrzeni zmiennych skutkuje innym drzewem), a także mniejsza dokładność względem nowszych metod (np. sieci neuronowych).

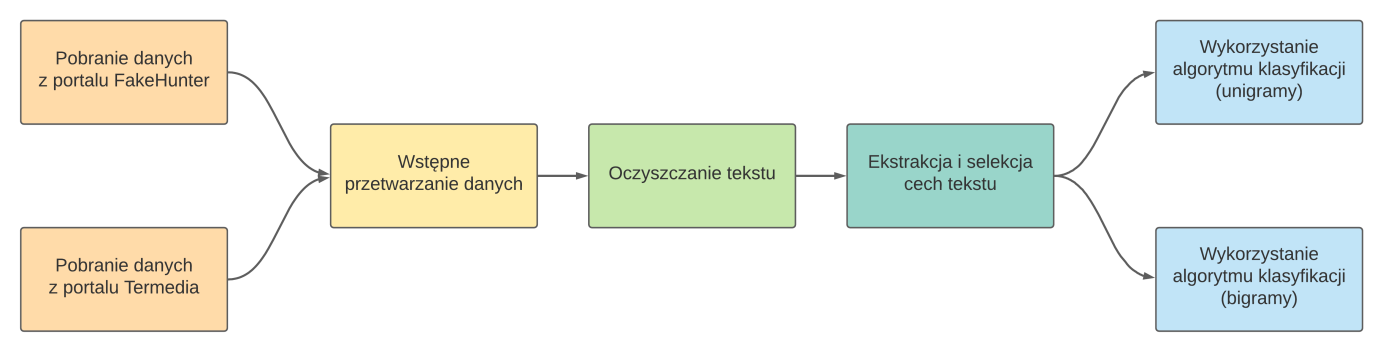
W celu poprawienia rezultatów, drzewa, z których składa się las losowy, tworzone są na podstawie wylosowanego ze zwracaniem podzbioru dostępnych danych, a podział w węzłach tych drzew odbywa się z wykorzystaniem pewnej wylosowanej liczby predyktorów – poprzez wybór najlepszego podziału dla tego podzbioru zmiennych objaśniających. W przypadku klasyfikacji prognozę algorytmu stanowi wynik głosowania wśród drzew (klasa wybrana przez większość), a przy wyznaczaniu wartości liczbowej zmiennej objaśnianej w regresji, średnia wyników uzyskanych przez poszczególne drzewa. [42]

W modelu lasu losowego z biblioteki scikit-learn, wykorzystanym w realizacji projektu, prognozę stanowi średnia predykcji wszystkich drzew, na podstawie której określana jest przynależność do klasy, a nie klasa wybrana przez większość. [43]

## Model klasyfikatora

Kompletny model klasyfikatora wykorzystany w niniejszej pracy, przedstawiony na Rysunku 3.6, składa się z następujących kroków:

1. Pobranie danych z portalu FakeHunter i Termedia (Rozdział 3.1).
2. Wstępne przetwarzanie danych (Rozdział 3.2.1).
3. Oczyszczanie tekstu (Rozdział 3.2.1).
4. Ekstrakcja i selekcja cech tekstu (Rozdział 3.2.2).
5. Wykorzystanie algorytmu klasyfikacji (Rozdział 3.3).



**Rysunek 3.15.** Model klasyfikatora wykorzystany do detekcji tzw. „fake newsów”.

Parametry algorytmów klasyfikacji dobierano za pomocą metody tzw. „grid search” z biblioteki scikit-learn, która dokonuje optymalizacji modelu na podstawie przeglądu siatki parametrów i wykorzystania mechanizmu sprawdzianu krzyżowego [44].

W przypadku naiwnego klasyfikatora bayesowskiego manipulowano wartością współczynnika , odpowiadającego za stopień wygładzania addytywnego ( oznacza brak wygładzania) [37], wybierając go ze zbioru .

W przypadku liniowej maszyny wektorów nośnych modyfikowano stałą regularyzacji (stopień regularyzacji jest odwrotnie proporcjonalny do ) [40], wybierając go ze zbioru .

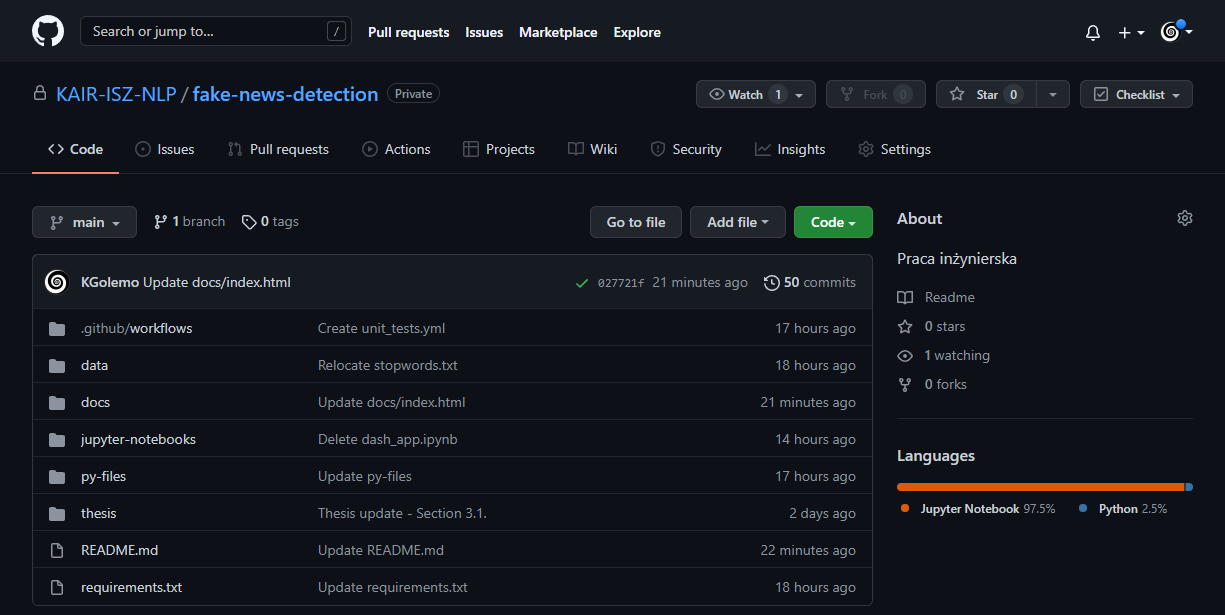
W przypadku lasu losowego zmieniano parametr (liczbę dostępnych drzew w lesie) [43], wybierając go ze zbioru .

Rezultaty ewaluacji modelów klasyfikatora zostały omówione w Rozdziale 4.2.

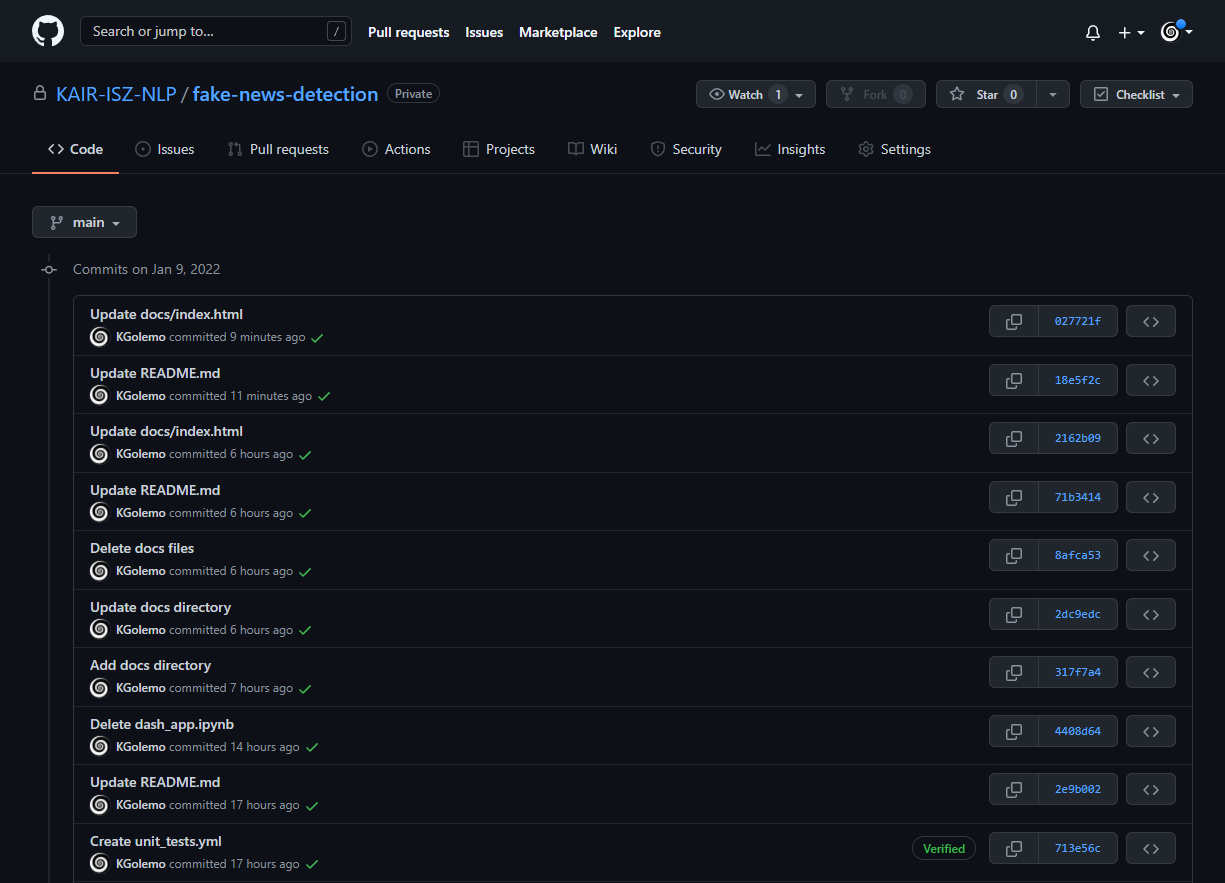
## Tworzenie oprogramowania

Projekt został zrealizowany z wykorzystaniem notatników Jupyter (ang. *Jupyter Notebooks*) [45], umożliwiających tworzenie kodu w języku Python w formie uruchamialnych dokumentów. Proces rozwoju oprogramowania przeprowadzony został w oparciu o system kontroli wersji Git, z wykorzystaniem założonego w serwisie GitHub repozytorium projektu [46]. Umożliwiło to m.in. śledzenie zmian w plikach, czy wykorzystanie mechanizmu ciągłej integracji (Rozdział 4.1).

Na Rysunku 3.16 przedstawiono panel główny repozytorium, z widocznym m.in. katalogiem *jupyter-notebooks* zawierającym wspomniane notatniki, czy *data*, w którym przechowywany jest zbiór danych. Lokalne zmiany nanoszone w trakcie tworzenia oprogramowania były każdorazowo przesyłane do repozytorium za pomocą tzw. „commitów”, których historia widoczna jest na Rysunku 3.17.



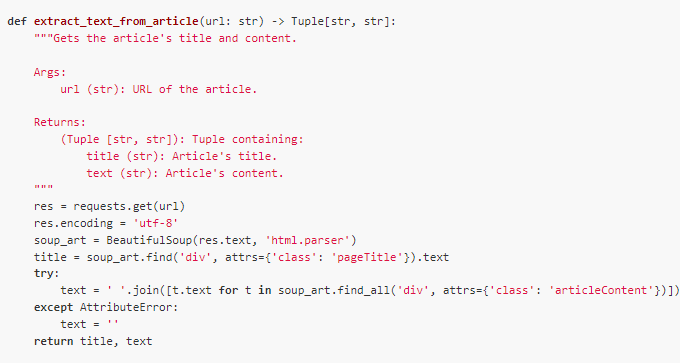
**Rysunek 3.16.** Panel główny repozytorium projektu z widocznym m.in. katalogiem *thesis* zawierającym niniejszą pracę oraz *data*, w którym przechowywany jest zbiór danych



**Rysunek 3.17.** Fragment historii tzw. „commitów”, czyli operacji przesłania zmian do repozytorium, z widocznymi zielonym znacznikami związanymi z mechanizmem ciągłej integracji (Rozdział 4.1)

## Dokumentacja

Dokumentacja opracowanego oprogramowania w formie interaktywnej witryny internetowej [47] została wygenerowana za pomocą narzędzia *pdoc3* [48] na podstawie dodanych do każdej zaimplementowanej funkcji przetwarzającej zbiór danych (Rozdział 3.2) tzw. „docstringów”. Elementy te, widoczne na Rysunku 3.18, stanowią opis realizowanego przez funkcję zadania, jej argumentów oraz zwracanych przez nią obiektów. Na Rysunku 3.19 przedstawiono fragment dokumentacji z widocznym panelem nawigacyjnym, umożliwiającym przemieszczanie się między opisami wszystkich funkcji.



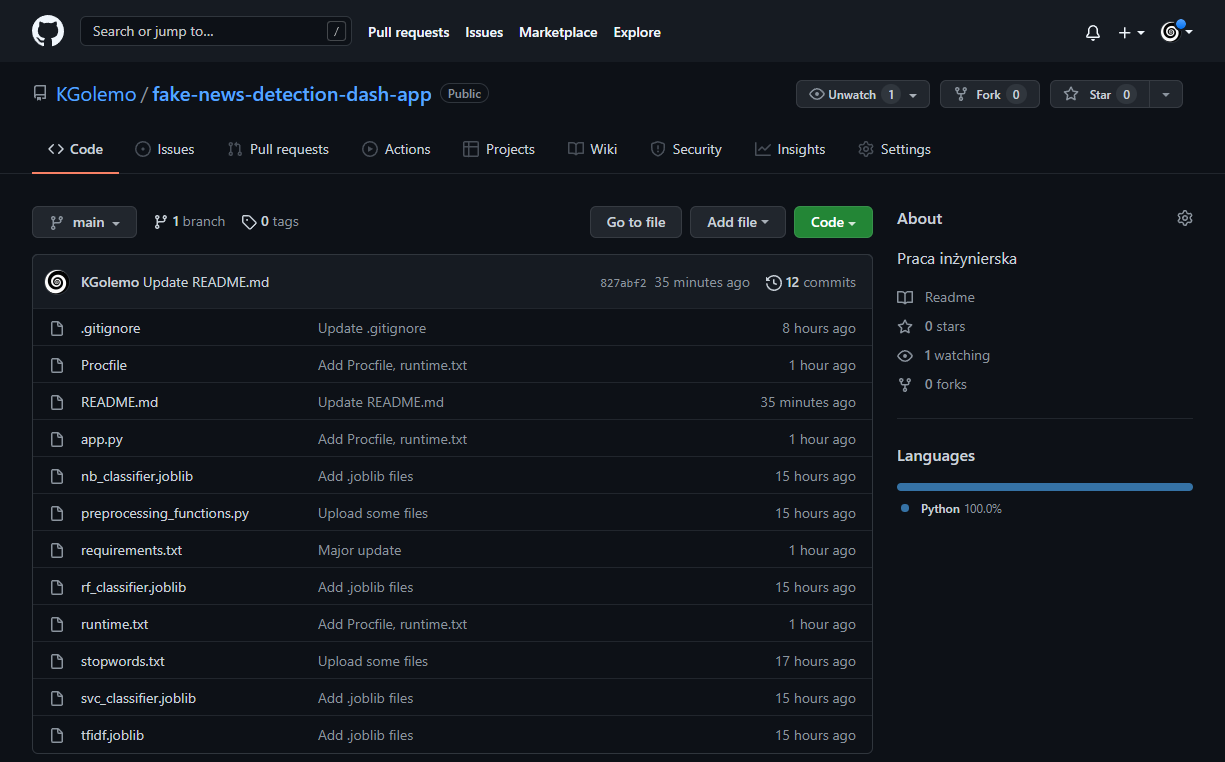
**Rysunek 3.18.** Przykładowy opis za pomocą tzw. „docstringa” funkcji odpowiadającej za wyodrębnienie tekstu z artykułu z portalu Termedia



**Rysunek 3.19.** Fragment dokumentacji projektu z widocznym po lewej stronie panelem nawigacyjnym, umożliwiającym przemieszczanie się pomiędzy opisami wszystkich funkcji

## Tworzenie aplikacji webowej

W celu stworzenia aplikacji umożliwiającej użytkownikowi zweryfikowanie wybranej wiadomości, konieczne było wyeksportowanie za pomocą narzędzia *Joblib* [49] obiektów związanych z ekstrakcją cech tekstu oraz wytrenowanych algorytmy klasyfikacji. Wszelkie pliki konieczne do działania aplikacji i modelu klasyfikatora zostały umieszczone w osobnym repozytorium [50], którego panel główny widoczny jest na Rysunku 3.20. Do zaprojektowania interfejsu wykorzystano pakiet *Dash* [51], umożliwiający tworzenie prostych witryn internetowych za pomocą języka Python, a w celu opublikowania aplikacji skorzystano z bezpłatnej platformy chmurowej *Heroku* [52].

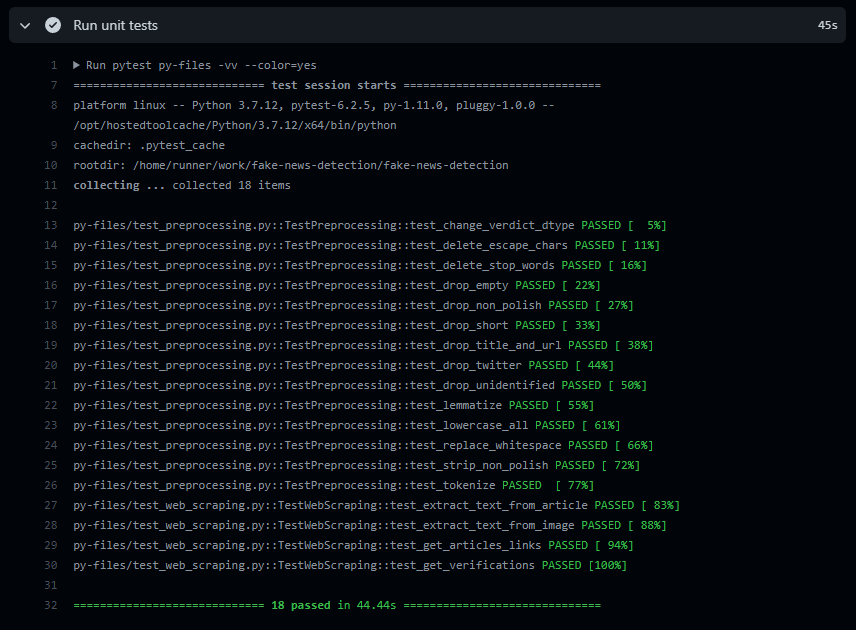


**Rysunek 3.20.** Panel główny repozytorium aplikacji z widocznymi m.in. plikami *.joblib* zawierającymi obiekty związane z modelem klasyfikatora

# Rezultaty

## Testy jednostkowe funkcji

Stworzone oprogramowanie zostało pokryte testami jednostkowymi *unittest* [53], weryfikującymi działanie wszystkich zaimplementowanych funkcji odpowiadających za przetwarzanie zbioru danych (Rozdział 3.2). Wykorzystano także mechanizm ciągłej integracji (ang. *Continuous Integration*, *CI*), polegający na weryfikowaniu za pomocą testów każdej aktualizacji projektu w repozytorium. Mechanizm ten zaimplementowano za pomocą dostępnych GitHub Actions, które korzystając z zamieszczonego w katalogu *workflows* pliku, przeprowadzają określone w nim operacje związane z ciągłą integracją. Rezultaty przykładowej weryfikacji przedstawiono na Rysunku 4.1.

****

**Rysunek 4.1.** Rezultat przykładowej weryfikacji przeprowadzonej przez GitHub Actions w ramach mechanizmu ciągłej integracji

## Ewaluacja algorytmów klasyfikacji

### Zastosowane metryki

W celu ewaluacji klasyfikacji przeprowadzonych z użyciem algorytmów wymienionych w Rozdziale 3.3, wykorzystano następujące metryki z biblioteki scikit-learn [54]:

* precyzję (ang. *precision*),
* czułość (ang. *recall*),
* współczynnik -score,
* dokładność (ang. *accuracy*),
* pole pod krzywą ROC (ang. *Receiver Operating Characteristic*).

Zgodnie z [55] w ewaluacji binarnych klasyfikacji wyróżnia się cztery rodzaje predykcji:

* przypadki prawdziwie pozytywne (ang. *true positive, tp*) – poprawne klasyfikacje pozytywnych przypadków (prawdziwe teksty zidentyfikowane jako prawdziwe),
* przypadki prawdziwie negatywne (ang. *true negative, tn*) – poprawne klasyfikacje negatywnych przypadków (fałszywe teksty zidentyfikowane jako fałszywe),
* przypadki fałszywie pozytywne (ang. *false positive, fp*) – niepoprawne klasyfikacje negatywnych przypadków (fałszywe teksty zidentyfikowane jako prawdziwe),
* przypadki fałszywie negatywne (ang. *false negative, fn*) – niepoprawne klasyfikacje pozytywnych przypadków (prawdziwe teksty zidentyfikowane jako fałszywe).

Precyzja określa jaką część wszystkich pozytywnych przypadków stanowią predykcje prawdziwie pozytywne [54]:

|  |  |  |
| --- | --- | --- |
|  |  | (4.1.1) |

Czułość stanowi stosunek prawdziwie pozytywnych predykcji do wszystkich pozytywnych przypadków [54]:

|  |  |  |
| --- | --- | --- |
|  |  | (4.1.2) |

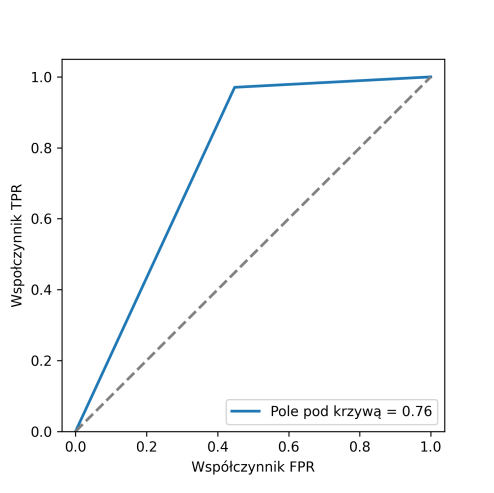
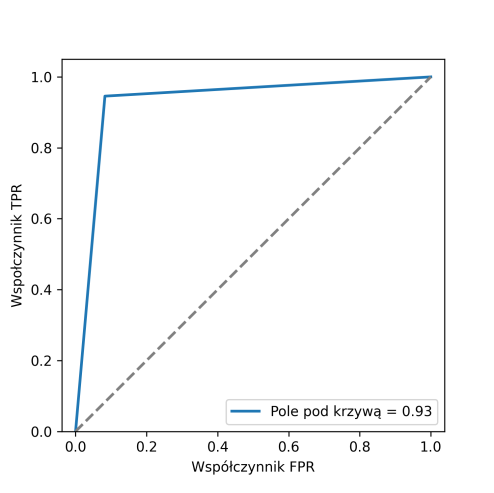
-score stanowi harmoniczną średnią precyzji i czułości [54]:

|  |  |  |
| --- | --- | --- |
|  |  | (4.1.3) |

Dokładność określa jaką część wszystkich predykcji stanowią poprawne klasyfikacje [54]:

|  |  |  |
| --- | --- | --- |
|  |  | (4.1.4) |

Krzywa ROC obrazuje zależność między tzw. współczynnikiem TPR (ang. *True Positive Rate*) – stosunkiem predykcji prawdziwie pozytywnych do wszystkich pozytywnych przypadków, a tzw. współczynnikiem FPR (ang. *False Positive* Rate) – stosunkiem predykcji fałszywie pozytywnych do wszystkich negatywnych przypadków. Jeżeli konieczne jest ocenienie jakości klasyfikacji za pomocą jednej metryki, pole pod tą krzywą jest jedną z najlepszych możliwości, ponieważ wskazuje ono jak dobrze klasy są separowane [55]. Na Rysunku 4.2 przedstawiono przykładowe krzywe ROC wraz z obliczonym polem pod nimi.



**Rysunek 4.2**. Przykładowe krzywe ROC z obliczonymi pod nimi polami

### Naiwny klasyfikator bayesowski

**Tabela 4.1.** Rezultaty ewaluacji klasyfikacji algorytmem NB przeprowadzonej na zbiorze testowym (unigramy) przy pomocy precyzji, czułości, współczynnika -score, dokładności i pola pod krzywą ROC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Liczba**  **cech** |  | **Precyzja []** | **Czułość []** | **Współczynnik -score []** | **Dokładność []** | **Pole pod**  **krzywą ROC** |
| 1000 | 0,1 | 85 | 96 | 90 | 87 | 0,82 |
| 3000 | 0,1 | 88 | 96 | 92 | 90 | 0,86 |
| 10000 | 0,03 | 93 | 95 | **94** | **92** | **0,90** |
| 27416 | 0,01 | 88 | 96 | 92 | 89 | 0,86 |

**Tabela 4.2.** Rezultaty ewaluacji klasyfikacji algorytmem NB przeprowadzonej na zbiorze testowym (bigramy)

przy pomocy precyzji, czułości, współczynnika -score, dokładności i pola pod krzywą ROC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Liczba**  **cech** |  | **Precyzja []** | **Czułość []** | **Współczynnik -score []** | **Dokładność []** | **Pole pod**  **krzywą ROC** |
| 1000 | 0,1 | 79 | 97 | 87 | 81 | 0,74 |
| 3000 | 0,1 | 80 | 97 | **88** | **83** | 0,76 |
| 10000 | 0,03 | 81 | 97 | **88** | **83** | **0,77** |
| 30000 | 0,01 | 81 | 97 | **88** | **83** | 0,76 |
| 100000 | 0,001 | 80 | 97 | **88** | 82 | 0,75 |
| 193552 | 0,3 | 76 | 98 | 86 | 78 | 0,69 |

W przypadku unigramów (Tabela 4.1) największą wartość współczynnika -score, dokładności i pola pod krzywą ROC uzyskano przy ograniczeniu liczby cech do 10 tysięcy.

W przypadku bigramów (Tabela 4.2), największą wartość współczynnika -score osiągnięto przy ograniczeniu liczby cech do 3 tysięcy, 10 tysięcy, 30 tysięcy i 100000 tysięcy, największą dokładność przy ograniczeniu do 3, 10 i 30 tysięcy, a największe pole pod krzywą ROC przy ograniczeniu do 10 tysięcy.

Ogółem dla naiwnego klasyfikatora bayesowskiego najlepsze rezultaty uzyskano przy ograniczeniu liczby cech do 10000 w przestrzeni opartej o unigramy. Dodatkowa analiza bigramów nie poprawiła jakości klasyfikacji, zmniejszając wartość pola pod krzywą ROC o ok. 17%.

### Liniowa maszyna wektorów nośnych

**Tabela 4.3.** Rezultaty ewaluacji klasyfikacji algorytmem LSVM przeprowadzonej na zbiorze testowym (unigramy) przy pomocy precyzji, czułości, współczynnika -score, dokładności i pola pod krzywą ROC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Liczba**  **cech** |  | **Precyzja []** | **Czułość []** | **Współczynnik -score []** | **Dokładność []** | **Pole pod**  **krzywą ROC** |
| 1000 | 0,1 | 91 | 94 | 93 | 90 | 0,88 |
| 3000 | 0,3 | 94 | 94 | 94 | 92 | 0,91 |
| 10000 | 0,3 | 96 | 94 | **95** | **94** | **0,94** |
| 27416 | 1 | 96 | 94 | **95** | **94** | 0,93 |

**Tabela 4.4.** Rezultaty ewaluacji klasyfikacji algorytmem LVSM przeprowadzonej na zbiorze testowym (bigramy) przy pomocy precyzji, czułości, współczynnika -score, dokładności i pola pod krzywą ROC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Liczba**  **cech** |  | **Precyzja []** | **Czułość []** | **Współczynnik -score []** | **Dokładność []** | **Pole pod**  **krzywą ROC** |
| 1000 | 0,3 | 90 | 92 | 91 | 88 | 0,86 |
| 3000 | 0,3 | 90 | 93 | 91 | 88 | 0,85 |
| 10000 | 1 | 91 | 94 | **92** | **90** | **0,88** |
| 30000 | 1 | 89 | 95 | **92** | 89 | 0,86 |
| 100000 | 1 | 85 | 96 | 91 | 87 | 0,82 |
| 193552 | 1 | 81 | 97 | 88 | 83 | 0,76 |

W przypadku unigramów (Tabela 4.3) największą wartość współczynnika -score i dokładności osiągnięto przy ograniczeniu liczby cech do 10 tysięcy i pozostawieniu ich domyślnej liczby, a największe pole pod krzywą ROC przy ograniczeniu liczby cech do 10 tysięcy.

W przypadku bigramów (Tabela 4.4), największą wartość współczynnika -score osiągnięto przy ograniczeniu liczby cech do 10 tysięcy i 30 tysięcy, a największą dokładność i pole pod krzywą ROC przy ograniczeniu do 10 tysięcy.

Ogółem dla maszyny wektorów nośnych najlepsze rezultaty uzyskano przy ograniczeniu liczby cech do 10000 w przestrzeni opartej o unigramy. Dodatkowa analiza bigramów nie poprawiła jakości klasyfikacji, zmniejszając wartość pola pod krzywą ROC o ok. 7%.

### Las losowy

**Tabela 4.5.** Rezultaty ewaluacji klasyfikacji przeprowadzonej algorytmem RF na zbiorze testowym (unigramy)

przy pomocy precyzji, czułości, współczynnika -score, dokładności i pola pod krzywą ROC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Liczba**  **cech** |  | **Precyzja []** | **Czułość []** | **Współczynnik -score []** | **Dokładność []** | **Pole pod**  **krzywą ROC** |
| 1000 | 1000 | 91 | 92 | **92** | **89** | 0,87 |
| 3000 | 1000 | 91 | 93 | **92** | **89** | **0,88** |
| 10000 | 300 | 91 | 93 | **92** | **89** | **0,88** |
| 27416 | 300 | 91 | 93 | **92** | **89** | 0,87 |

**Tabela 4.6.** Rezultaty ewaluacji klasyfikacji algorytmem RF przeprowadzonej na zbiorze testowym (bigramy)

przy pomocy precyzji, czułości, współczynnika -score, dokładności i pola pod krzywą ROC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Liczba**  **cech** |  | **Precyzja []** | **Czułość []** | **Współczynnik -score []** | **Dokładność []** | **Pole pod**  **krzywą ROC** |
| 1000 | 1000 | 93 | 88 | 90 | 88 | 0,87 |
| 3000 | 300 | 94 | 87 | 90 | 88 | 0,88 |
| 10000 | 1000 | 95 | 88 | 91 | 89 | 0,90 |
| 30000 | 100 | 97 | 85 | 91 | 88 | 0,90 |
| 100000 | 300 | 99 | 83 | 91 | 89 | **0,91** |
| 193552 | 100 | 98 | 86 | **92** | **90** | **0,91** |

W przypadku unigramów (Tabela 4.5) wartość współczynnika -score i dokładności przyjmowała te same wartości dla każdej liczby cech, a największe pole pod krzywą ROC uzyskano przy ograniczeniu liczby cech do 3 i 10 tysięcy.

W przypadku bigramów (Tabela 4.6), największą wartość współczynnika -score i dokładność osiągnięto dla domyślnej liczby cech, a największe pole pod krzywą ROC przy ograniczeniu do 100 tysięcy i pozostawieniu ich domyślnej liczby.

Ogółem dla lasu losowego najlepsze rezultaty uzyskano przy domyślnej liczby cech w przestrzeni opartej o bigramy. Dodatkowa analiza bigramów nieznacznie poprawiła jakości klasyfikacji, zwiększając wartość pola pod krzywą ROC o ok. 3%.

## Aplikacja webowa

Stworzona aplikacja [56] umożliwia użytkownikowi zweryfikowanie wybranej wiadomości poprzez wklejenie jej treści do przeznaczonego do tego pola, a następnie przyciśnięcie obiektu rozpoczynającego proces. Elementy te przedstawiono na Rysunku 4.3.

Po uruchomieniu analizy przeprowadzane są następujące operacje:

1. Oczyszczenie tekstu za pomocą funkcji przetwarzających dane (Rozdział 3.2).
2. Przeniesienie oczyszczonego tekstu do przestrzeni cech za pomocą obiektu wektoryzującego odtworzonego z pliku *.joblib*. (Rozdział 3.2)
3. Wyznaczenie predykcji za pomocą wytrenowanych algorytmów klasyfikacyjnych odtworzonych z pliku *.joblib*. (Rozdział 3.3)

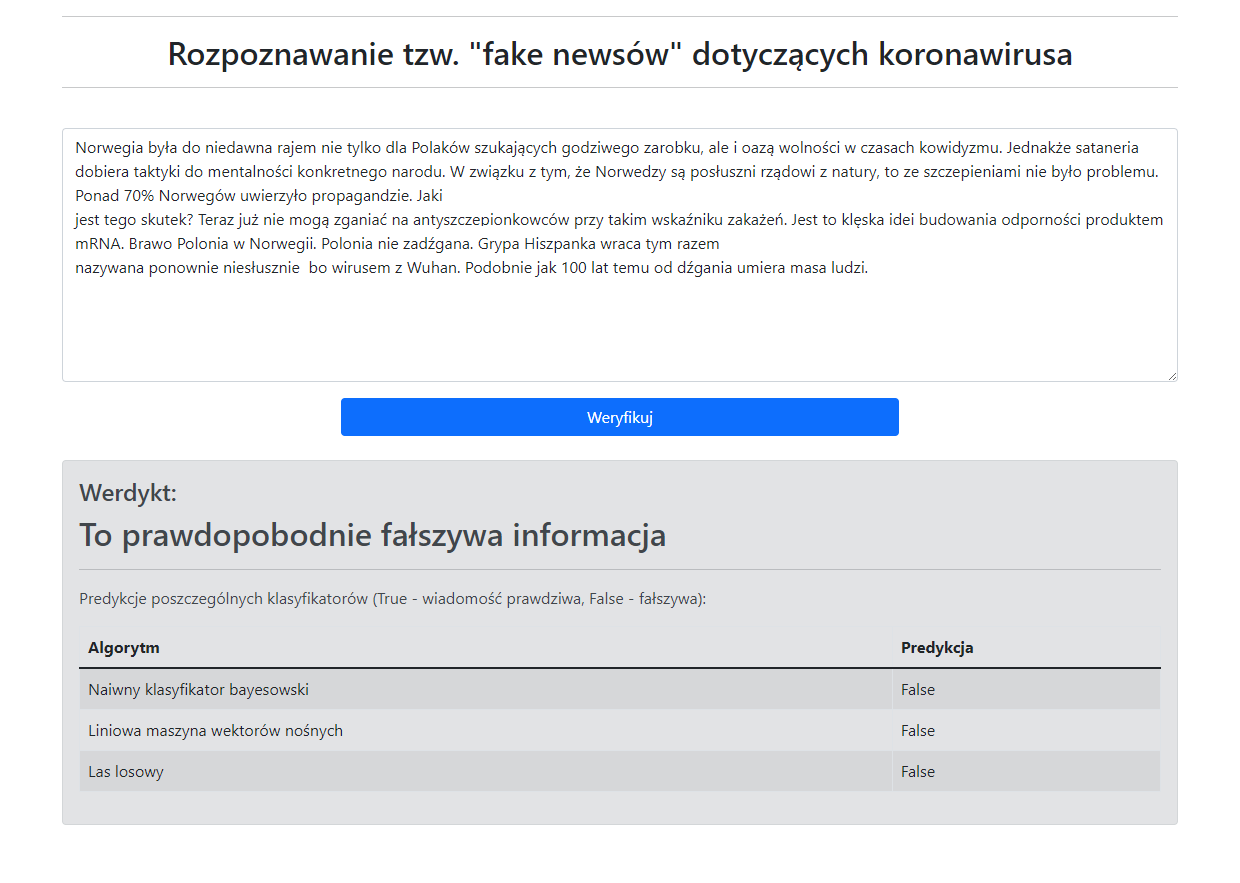
Końcowy werdykt wyznaczany jest w następujący sposób:

* informacja jest uznawana za fałszywą, jeśli przynajmniej jeden z klasyfikatorów uznał ją za fałszywą,
* informacja jest uznawana za prawdziwą, jeśli wszystkie klasyfikatory uznały ją za prawdziwą.

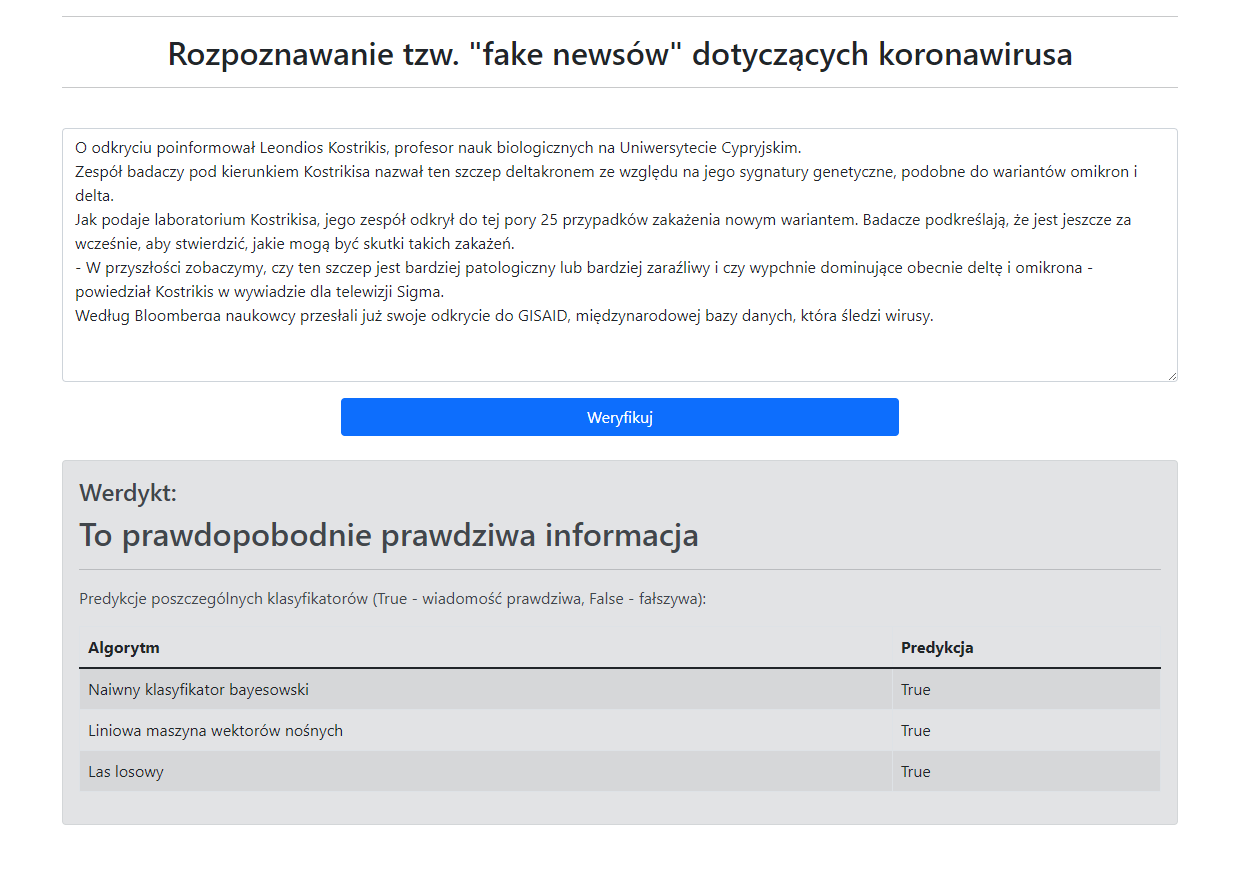
Raport z analizy zawiera dodatkowo informacje o wynikach predykcji poszczególnych algorytmów. Przykładowe weryfikacje zostały przestawione na Rysunku 4.4 i 4.5.



**Rysunek 4.3.** Domyślny wygląd aplikacji z widocznym polem do wklejenia treści wiadomości oraz przyciskiem rozpoczynającym proces weryfikacji



**Rysunek 4.4.** Rezultat weryfikacji w przypadku fałszywej informacji



**Rysunek 4.5.** Rezultat weryfikacji w przypadku prawdziwej informacji

# Podsumowanie

## Analiza wyników klasyfikacji

Poniżej zamieszczono najlepsze pod względem zastosowanych metryk wyniki uzyskane przez wszystkie ewaluowane algorytmy dla uni- i bigramowych przestrzeni cech (Rozdział 4.2). Porównano wartości współczynników -score, poziomy dokładności oraz pola pod krzywymi ROC.

**Tabela 5.1.** Rezultaty ewaluacji klasyfikacji przeprowadzonej na zbiorze testowym (unigramy)

przy pomocy współczynnika -score, dokładności i pola pod krzywą ROC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorytm klasyfikacji** | **Liczba**  **cech** | **Współczynnik -score []** | **Dokładność []** | **Pole pod**  **krzywą ROC** |
| NB | 10000 | 94 | 92 | 0,90 |
| LSVM | 10000 | **95** | **94** | **0,94** |
| RF | 10000 | 92 | 89 | 0,88 |

**Tabela 5.2.** Rezultaty ewaluacji klasyfikacji przeprowadzonej na zbiorze testowym (bigramy)

przy pomocy współczynnika -score, dokładności i pola pod krzywą ROC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorytm klasyfikacji** | **Liczba**  **cech** | **Współczynnik -score []** | **Dokładność []** | **Pole pod**  **krzywą ROC** |
| NB | 10000 | 88 | 83 | 0,77 |
| LSVM | 10000 | 92 | 90 | 0,88 |
| RF | 193552 | **92** | **90** | **0,91** |

W przypadku unigramów (Tabela 5.1) dla wszystkich algorytmów najlepsze rezultaty uzyskano przy ograniczeniu liczby cech do 10 tysięcy. Największe wartości współczynnika -score, dokładności oraz pola pod krzywą ROC otrzymano dla klasyfikatora LVSM (kolejno 95%, 94% i 0,94), a najmniejsze dla RF (kolejno 92%, 89%, 0,88).

W przypadku bigramów (Tabela 5.2) dla algorytmów NB i LVSM najlepsze rezultaty uzyskano przy ograniczeniu liczby cech do 10 tysięcy, a dla klasyfikatora RF przy ich pełnym zbiorze. Największe wartości współczynnika -score, dokładności oraz pola pod krzywą ROC otrzymano dla klasyfikatora RF (kolejno 92%, 90% i 0,91), a najmniejsze dla NB (kolejno 88%, 83%, 0,77).

Ogółem najlepsze rezultaty uzyskano przy zastosowaniu ograniczonej do 10 tysięcy cech przestrzeni unigramów i algorytmu LVSM – wartości współczynnika -score, dokładności oraz pola pod krzywą ROC wyniosły kolejno 95%, 94% i 0,94.

## Wnioski

Przeprowadzone badanie literaturowe umożliwiło wybór metod ekstrakcji cech tekstu oraz algorytmów odpowiednich do tego typu problemu klasyfikacji. Zaproponowany sposób

Stworzone oprogramowanie rozpoznaje tzw. „fake newsy” dotyczące koronawirusa, poprzez aplikację umożliwiającą użytkownikowi zweryfikowanie wybranej wiadomości.

## Kierunki dalszego rozwoju

W zaproponowanym sposobie pobierania wiadomości stanowiących tzw. „fake newsy” za pomocą oprogramowania rozpoznającego znaki

są kilkaset, tamte są pod tysiąc, słabo, wada, pobieranie danych z więcej niż jednego portalu, CNN, metody ekstrakcji i selekcji

# Bibliografia

x

|  |  |
| --- | --- |
| 1. | Gelfert A. Fake News: A Definition. Informal Logic. 2018 marzec: p. 84-117. |
| 2. | Molina MD, Sundar SS, Quang Le T, Lee D. “Fake News” Is Not Simply False Information: A Concept Explication and Taxonomy of Online Content. American Behavioral Scientist. 2019 październik. |
| 3. | FakeHunter. [Online]. [dostęp: 1 styczeń 2022]. <https://fakehunter.pap.pl>. |
| 4. | FakeNews.pl. [Online]. [dostęp: 1 styczeń 2022]. <https://fakenews.pl/>. |
| 5. | AntyFAKE. [Online]. [dostęp: 1 styczeń 2022]. <https://www.antyfake.pl/>. |
| 6. | Demagog. [Online]. [dostęp: 1 styczeń 2022]. <https://demagog.org.pl/>. |
| 7. | Konkret24. [Online]. [dostęp: 1 styczeń 2022]. <https://konkret24.tvn24.pl/>. |
| 8. | Molnar C. Interpretable machine learning: lulu.com; 2020. |
| 9. | Chowdhury G. Natural language processing. Annual Review of Information Science and Technology. 2005 styczeń: p. 51-89. |
| 10. | Khurana D, Koli A, Khatter K, Singh S. Natural Language Processing: State of Art, Current Trends and Challenges. 2017. |
| 11. | Jurafsky D, Martin J. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd ed.: Prentice Hall; 2008. |
| 12. | Allahyari M, Pouriyeh S, Assefi M, Safaei S. A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques. 2017 lipiec. |
| 13. | Granik M, Mesyura V. Fake news detection using naive Bayes classifier. In IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON); 2017; Kijów. p. 900-903. |
| 14. | Ahmed H, Traore I, Saad S. Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments; 2017; Vancouver. p. 127-138. |
| 15. | Reis JCS, Correia A, Murai F, Veloso A, Benevenuto F. Supervised Learning for Fake News Detection. IEEE Intelligent Systems. 2019 marzec-kwiecień: p. 76-81. |
| 16. | Dokumentacja algorytmu XGBoost. [Online]. [dostęp: 5 styczeń 2022]. <https://xgboost.readthedocs.io/en/latest/index.html>. |
| 17. | Vogel I, Jiang P. Fake News Detection with the New German Dataset “GermanFakeNC”. In Digital Libraries for Open Knowledge; 2019; Oslo. p. 288-295. |
| 18. | Albawi S, Mohammed TA, Al-Zawi S. Understanding of a convolutional neural network. In International Conference on Engineering and Technology (ICET); 2017. p. 1-6. |
| 19. | Sidorov G, Moreno J, Adorno HG, Posadas-Durán JP. Detection of fake news in a new corpus for the Spanish language. Journal of Intelligent & Fuzzy Systems. 2019 maj; 36(5): p. 4869-4876. |
| 20. | Cavnar WB, Trenkle JM. N-gram-based text categorization. In Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval; 1994. |
| 21. | Schapire RE. The Boosting Approach to Machine Learning: An Overview. In Denison DD, Holmes CC, Hansen MH, Mallick B, Yu B. Nonlinear Estimation and Classification. Lecture Notes in Statistics. Nowy Jork: Springer; 2003. p. 149-171. |
| 22. | van der Linden S, Roozenbeek J, Compton J. Inoculating Against Fake News About COVID-19. Frontiers in Psychology. 2020 październik. |
| 23. | Diouf R, Sall O, Birregah B, Bousso M, Sarr EN, Bousso M, et al. Web Scraping: State-of-the-Art and Areas of Application. In IEEE International Conference on Big Data; 2019; Los Angeles. p. 6040-6042. |
| 24. | Dokumentacja biblioteki Requests. [Online]. [dostęp: 5 styczeń 2022]. |
| 25. | Dokumentacja biblioteki Beautiful Soup. [Online]. [dostęp: 5 styczeń 2022]. <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>. |
| 26. | Raport 1 z portalu FakeHunter. [Online]. [dostęp: 3 styczeń 2022]. <https://fakehunter.pap.pl/raport/2143bb38-fa97-4a76-a9f1-40a322d5e29b>. |
| 27. | Dokumentacja biblioteki EasyOCR. [Online]. [dostęp: 5 styczeń 2022]. <https://jaided.ai/easyocr/documentation/>. |
| 28. | Raport 2 z portalu FakeHunter. [Online]. [dostęp: 5 styczeń 2022]. <https://fakehunter.pap.pl/raport/eafff780-4b0e-441a-8329-cc26845c5279>. |
| 29. | Raport 3 z portalu FakeHunter. [Online]. [dostęp: 5 styczeń 2022]. <https://fakehunter.pap.pl/raport/fa241a92-7327-49c6-8b63-621264a80956>. |
| 30. | Termedia. [Online]. [dostęp: 12 grudzień 2021]. <https://www.termedia.pl>. |
| 31. | Dokumentacja biblioteki polyglot. [Online]. [dostęp: 5 styczeń 2022]. <https://polyglot.readthedocs.io/en/latest/>. |
| 32. | Kieraś W, Woliński M. Morfeusz 2 – analizator i generator fleksyjny dla języka polskiego. Język Polski. 2017: p. 75-83. |
| 33. | Dokumentacja metody TF-IDF (scikit-learn). [Online]. [dostęp: 5 styczeń 2022]. <https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction>. |
| 34. | McCallum A, Nigam K. A Comparison of Event Models for Naive Bayes Text Classification. 2001 maj. |
| 35. | Zhang H. The Optimality of Naive Bayes. In Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference; 2004; Floryda. |
| 36. | Raghavan P, Schütze H, Manning C. Introduction to Information Retrieval Cambridge: Cambridge University Press; 2009. |
| 37. | Dokumentacja naiwnego klasyfikatora bayesowskiego (scikit-learn). [Online]. [dostęp: 18 grudzień 2021]. <https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes>. |
| 38. | Boser B, Guyon I, Vapnik V. A Training Algorithm for Optimal Margin Classifier. Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory. 1992: p. 144-152. |
| 39. | Cortes C, Vapnik V. Support-vector networks. Machine Learning. 1995: p. 273-297. |
| 40. | Dokumentacja liniowej maszyny wektorów nośnych (scikit-learn). [Online]. [dostęp: 20 grudzień 2021]. <https://scikit-learn.org/stable/modules/svm.html#id15>. |
| 41. | Cutler A, Cutler DR, Stevens JR. Random Forests. Machine Learning. 2011 styczeń: p. 157-176. |
| 42. | Breiman L. Random Forests. Machine Learning. 2001 październik: p. 5-32. |
| 43. | Dokumentacja algorytmu lasu losowego (scikit-learn). [Online]. [dostęp: 24 grudzień 2021]. <https://scikit-learn.org/stable/modules/ensemble.html#forest>. |
| 44. | Dokumentacja metody grid search (scikit-learn). [Online]. [dostęp: 5 styczeń 2022]. <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html>. |
| 45. | Dokumentacja notatników Jupyter. [Online]. [dostęp: 8 styczeń 2022]. <https://docs.jupyter.org/en/latest/>. |
| 46. | Repozytorium GitHub projektu. [Online]. [dostęp: 8 styczeń 2022]. <https://github.com/KAIR-ISZ-NLP/fake-news-detection>. |
| 47. | Dokumentacja projektu. [Online]. [dostęp: 8 styczeń 2022]. <https://kair-isz-nlp.github.io/fake-news-detection/>. |
| 48. | Dokumentacja PEP 257. [Online]. [dostęp: 8 styczeń 2022]. <https://www.python.org/dev/peps/pep-0257/>. |
| 49. | Dokumentacja pakietu Joblib. [Online]. [dostęp: 8 styczeń 2022]. <https://joblib.readthedocs.io/en/latest/>. |
| 50. | Repozytorium aplikacji. [Online]. [dostęp: 8 styczeń 2022]. <https://github.com/KGolemo/fake-news-detection-dash-app>. |
| 51. | Dokumentacja pakietu Dash. [Online]. [dostęp: 8 styczen 2022]. <https://dash.plotly.com/>. |
| 52. | Dokumentacja platformy Heroku. [Online]. [dostęp: 8 styczeń 2022]. <https://devcenter.heroku.com/categories/reference>. |
| 53. | Dokumentacja unittest. [Online]. [dostęp: 8 styczeń 2022]. <https://docs.python.org/3/library/unittest.html>. |
| 54. | Dokumentacja metryk (scikit-learn). [Online]. [dostęp: 5 styczeń 2022]. <https://scikit-learn.org/stable/modules/model_evaluation.html>. |
| 55. | Bradley AP. The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern recognition. 1997: p. 1145-1159. |
| 56. | Aplikacja. [Online]. [dostęp: 8 styczeń 2022]. <https://fake-news-detection-dash-app.herokuapp.com/>. |
| 57. | Dokumentacja generatora pdoc3. [Online]. [dostęp: 8 styczeń 2022]. <https://pdoc3.github.io/pdoc/>. |

x