UNIWERSYTET IM. ADAMA MICKIEWICZA
WYDZIAŁ MATEMATYKI I INFORMATYKI

Krzysztof Joachimiak
nr albumu: s412778

Zastosowanie technik uczenia maszynowego w przetwarzaniu języka naturalnego

Applying Machine Learning to Natural Language Processing

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
prof. UAM dr hab. Krzysztof Jassem

Poznań 2016

Poznań, czerwiec 2016

OŚWIADCZENIE

Ja, niżej podpisany Krzysztof Joachimiak, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt: *Zastosowanie technik uczenia maszynowego w przetwarzaniu języka naturalnego* napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

Wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM oraz na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich.

.....

podpis

STRESZCZENIE

Niniejsza praca prezentuje wyniki projektu związanego z zastosowaniem uczenia maszynowego w wykrywaniu obraźliwego języka w Internecie. Początkowe rozdziały prezentują wiedzę teoretyczną z zakresu przetwarzania języka naturalnego oraz uczenia maszynowego, omówiono ponadto wybrane prace dotyczące wykrywania obraźliwego języka. Ostatni rozdział przedstawia rezultaty osiągnięte w ramach projektu: oznaczony korpus komentarzy internetowych w języku polskim, autorskie narzędzia do przetwarzania języka naturalnego w języku programowania Python oraz system umożliwiający automatyczne cenzurowanie obraźliwych komentarzy na portalu Wirtualna Polska za pośrednictwem wtyczki do przeglądarki Google Chrome.

SŁOWA KLUCZOWE

przetwarzanie języka naturalnego, uczenie maszynowe, wykrywanie wypowiedzi obraźliwych, korpus obraźliwego języka, narzędzia do przetwarzania języka polskiego

ABSTRACT

This thesis presents results of the project, that relates to applying machine learning to the detection of the offensive language in the Internet. First chapters introduce teorethical background in the field of natural language processing and machine learning and discuss selected papers relating to offensive language detection. The last chapter presents the results achieved in the project: labelled corpus of Internet comments in the Polish language, author's tools for natural language processing in the Python programming language and the system that facilitates offensive comments censoring on Wirtualna Polska website using a Google Chrome plugin.

KEYWORDS

natural language processing, machine learning, offensive laguage detection, offensive language corpus, tools for the processing of the Polish language

Spis treści

ROZDZIAŁ 1

Wstęp

Wśród głównych problemów korzystania z otwartego Internetu wymienić można częste pojawianie się obraźliwych komentarzy, potocznie określanych mianem *hejtu*. Pozorna anonimowość internautów oraz łatwość publicznego wyrażania swoich opinii sprawiają, że wielu użytkowników Internetu przejawia skłonność do wyrażania opinii skrajnych, niejednokrotnie przekraczających granice kultury czy dobrego smaku. Duża liczba komentarzy utrudnia ich ręczną moderację przez administratorów stron, same komentarze bywają natomiast wykorzystywane do manipulowania opinią publiczną.

Dzięki narzędziom przetwarzania języka naturalnego oraz uczenia maszynowego możliwe jest stworzenie systemu, który automatycznie będzie w stanie wykrywać wypowiedzi mogące zostać uznane za obraźliwe. System taki mógłby znaleźć dwa podstawowe zastosowania: automatyczne cenzurowanie wypowiedzi obraźliwych oraz użycie w badaniach socjologicznych dotyczących Internetu (np. w celu ujawnienia zaplanowanych akcji dyskredytowania jakiejś osoby). W pracy przedstawione zostały wybrane metody uczenia maszynowego oraz zagadnienia związane z przetwarzaniem języka naturalnego. W ramach projektu magisterskiego zgromadzono i oznaczono korpus internetowych komentarzy z przeznaczeniem wykorzystania ich do treningu klasyfikatorów wyspecjalizowanych w wykrywaniu wypowiedzi obraźliwych. Stworzono również pomocnicze narzędzia do przetwarzania wypowiedzi w języku polskim. Przeprowadzono serię eksperymentów mających na celu porównanie różnych metod wektorowej reprezentacji tekstu oraz algorytmów uczenia maszynowego. Ostatecznym wynikiem projektu prezentowanego w niniejszej pracy jest system, który dzięki wtyczce do przeglądarki Google Chrome umożliwia wykrywanie i cenzurowanie obraźliwych wypowiedzi na portalu Wirtualna Polska.

ROZDZIAŁ 2

Uczenie maszynowe

2.1 Typy uczenia maszynowego

2.1.1 Uczenie nadzorowane

Uczenie nadzorowane (ang. *supervised learning*) polega na trenowaniu systemu na podstawie przykładów oznaczonych pewnymi etykietami (problem klasyfikacji) lub liczbami (regresja) [9]. Na wejściu systemu dostarczany jest zbiór uczący $\langle \mathbf{x}, \mathbf{y} \rangle$ [17], gdzie $\mathbf{x}$ oznacza zbiór wektorów opisujących przykłady uczące, natomiast $\mathbf{y}$ to zbiór etykiet (lub liczb) przypisanych kolejnym wektorom. Celem procesu uczenia jest znalezienie najlepszego przybliżenia funkcji opisanej w równaniu 2.1,

$$f(x_i) = y_i \tag{2.1}$$

gdzie x_i, y_i to odpowiednio i-ty element zbioru $\mathbf{x}$ i i-ty element zbioru $\mathbf{y}$ (zgodnie w przyjętą tutaj konwencją, i-ty przykład ze zbioru uczącego opisany etykietą będziemy oznaczać jako $\langle x_i, y_i \rangle$). Znalezienie takiej funkcji umożliwia nam przewidywanie, jaką etykietę należy przypisać nowemu przykładowi $\overline{x}$, który nie został uwzględniony na etapie uczenia [7]. Uczynione powyżej rozróżnienie klasyfikacji i regresji odnosi się do charakterystyki zbioru, do którego należą wszystkie możliwe wartości przyjmowane przez y_i.

- **Klasyfikacja**
  Każdy element y_i przyjmuje wartości należące do skończonego i przeliczalnego zbioru.

- **Regresja**
  Wartości przyjmowane przez y_i należą do przestrzeni ciągłej.

2.1.2 Uczenie nienadzorowane

W uczeniu nienadzorowanym (ang. *unsupervised learning*) system otrzymuje na wejściu zbiór uczący $\langle \mathbf{x} \rangle$ nieoznaczony żadnymi etykietami bądź liczbami (brak zbioru $\mathbf{y}$). Zadaniem stawianym w problemie uczenia nienadzorowanego jest znalezienie pewnego wzorca kryjącego się za pozornie niestrukturalizowanym zbiorem danych [9].

2.1.3 Uczenie półnadzorowane

Uczenie półnadzorowane (ang. *semi-supervised learning*) jest problemem znajdującym się niejako pośrodku między uczeniem nadzorowanym i nienadzorowanym. Systemowi dostarczany jest na wejściu zbiór uczący, który można symbolicznie zapisać jako $\langle \mathbf{x}, \mathbf{y}, \widetilde{\mathbf{x}} \rangle$, gdzie $\mathbf{x}$ jest zbiorem przykładów oznaczonych a $\mathbf{y}$ zbiorem odpowiadających mu etykiet, natomiast $\widetilde{\mathbf{x}}$ to zbiór przykładów nieposiadających etykiet.

2.2 Wektorowa reprezentacja tekstu

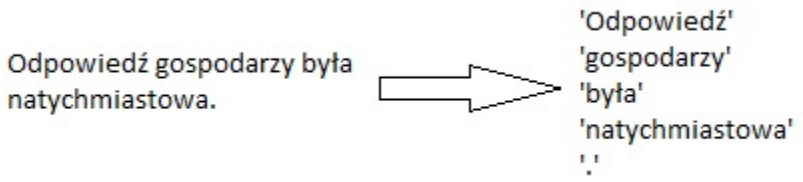
Aby możliwe było przeprowadzenie procesu uczenia na danych tekstowych, musimy reprezentować dokumenty wejściowe za pomocą wektorów w określonej przestrzeni cech. Dobór tych cech oraz sposób ich reprezentacji wchodzą w skład takich zagadnień uczenia maszynowego jak *feature engineering* czy *feature extraction*. Sam sposób reprezentacji możemy określić mianem modelu języka.

2.2.1 Wstępne przetwarzanie tekstu

Wprowadźmy oznaczenie d_i jako symboliczny zapis i-tego dokumentu tekstowego. Dokument taki możemy przekształcić do postaci pewnego multizbioru termów, aby ostatecznie przekształcić go w wektor liczb. Podczas transformacji tekst może zostać poddany następującym operacjom [39].

- **Tokenizacja**
  Tokenizacja jest rozbiciem tekstu na tokeny, czyli pojedyncze słowa lub wyrażenia.

Tokenami mogą być również znaki przestankowe. Przykład tokenizacji przedstawiono na rysunku 2.1. Wykonanie tokenizacji umożliwia przeprowadzenie dalszych operacji takich jak lematyzacja, rdzeniowanie czy oznaczanie części mowy.



Rysunek 2.1: Przykładowa tokenizacja

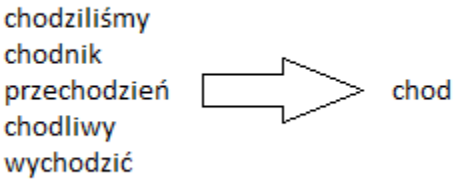
- **Lematyzacja**

  Lematyzacja polega na sprowadzaniu wyrazów do ich formy podstawowej zwanej także *lemmą* lub *lematem*.W lematyzacji zdarzają się przypadki nietrywialne – kilka ich przykładów (pochodzących z ręcznej lematyzacji słów w Korpusie Języka Polskiego, dalej jako KJP) przedstawiono w tabeli 2.1.

Przykład	**Opis**
***anglo**-amerykański*	Można zlematyzować jako: - anglosaski - angielski W KJP: angielski
*Podczas **patroszenia** dzika.*	Można zlematyzować jako: - patroszyć - patroszenie W KJP: patroszyć
*Co ty? **Hajdegera** się naczytałeś?*	Niestandardowy zapis nazwiska Heidegger. W KJP jako: Hajdeger.
***Większe** wątpliwości mają...*	Można zlematyzować jako: - wielki - duży

Tabela 2.1: Nietrywialne problemy lematyzacji [36]

- **Rdzeniowanie (ang. *stemmming*)**
  Rdzeniowanie polega na wydzielaniu tematu wyrazu poprzez usunięcie afiksów [26]. W praktyce oznacza to, że jesteśmy w stanie reprezentować całą rodzinę wyrazów za pomocą jednego termu. Przykład rdzeniowania przedstawiono na rysunku 2.2.



Rysunek 2.2: Przykład rdzeniowania dokonanego na słowach należących do jednej rodziny wyrazowej

- **Oznaczanie części mowy (ang. *Part-of-Speech tagging, POS-Tagging*)**
  Oznaczanie części mowy do polega na określeniu, jaką częścią mowy jest dany wyraz. W procesie POS-taggingu możemy otrzymać również dodatkowe informacje morfologiczne takie jak liczba, przypadek itp. (dla odmiennych części mowy). Przykłady możliwych sposobów przeprowadzenia POS-taggingu przedstawia rysunek 2.3.

Rysunek 2.3: Różne możliwości oznaczenia części mowy dla jednego słowa

2.2.2 Sposoby wektorowej reprezentacji tekstu

2.2.2.1 Bag-of-words

W podejściu Bag-of-words (BOW) każdy dokument przekształcany jest na wektor o długości równej liczbie słów zawartych w pewnym słowniku. Słownik zwykle jest zbiorem unikalnych termów (słów) występujących w całym zbiorze dokumentów. Najczęściej nie są to wszystkie słowa, gdyż pomija się tzw. słowa przestankowe (ang. *stopwords*). Za słowa przestankowe uważa się wyrazy, które występują często, jednak nie są nośnikiem zbyt dużej ilości informacji (nie mają dużego wpływu na wydźwięk dokumentu). Przykładową listę słów przestankowych dla języka polskiego przedstawiona rysunku 2.4. Warto przy tym zwrócić uwagę, że na liście tej znajduje się słowo „nie", które, choć często używane, ma jednak istotny wpływ na znaczenie dokumentu.

a, aby, ach, acz, aczkolwiek, aj, albo, ale, ależ, ani, aż, bardziej, bardzo, bo, bowiem, by, byli, bynajmniej, być, był, była, było, były, będzie, będą, cali, cała, cały, ci, cię, ciebie, co, cokolwiek, coś, czasami, czasem, czemu, czy, czyli, daleko, dla, dlaczego, dlatego, do, dobrze, dokąd, dość, dużo, dwa, dwaj, dwie, dwoje, dziś, dzisiaj, gdy, gdyby, gdyż, gdzie, gdziekolwiek, gdzieś, i, ich, ile, im, inna, inne, inny, innych, iż, ja, ją, jak, jakaś, jakby, jaki, jakichś, jakie, jakiś, jakiż, jakkolwiek, jako, jakoś, je, jeden, jedna, jedno, jednak, jednakże, jego, jej, jemu, jest, jestem, jeszcze, jeśli, jeżeli, już, ją, każdy, kiedy, kilka, kimś, kto, ktokolwiek, ktoś, która, które, którego, której, który, których, którym, którzy, ku, lat, lecz, lub, ma, mają, mało, mam, mi, mimo, między, mną, mnie, mogą, moi, moim, moja, moje, może, możliwe, można, mój, mu, musi, my, na, nad, nam, nami, nas, nasi, nasz, nasza, nasze, naszego, naszych, natomiast, natychmiast, nawet, nią, nic, nich, nie, niech, niego, niej, niemu, nigdy, nim, nimi, niż, no, o, obok, od, około, on, ona, one, oni, ono, oraz, oto, owszem, pan, pana, pani, po, pod, podczas, pomimo, ponad, ponieważ, powinien, powinna, powinni, powinno, poza, prawie, przecież, przed, przede, przedtem, przez, przy, roku, również, sama, są, się, skąd, sobie, sobą, sposób, swoje, ta, tak, taka, taki, takie, także, tam, te, tego, tej, temu, ten, teraz, też, to, tobą, tobie, toteż, trzeba, tu, tutaj, twoi, twoim, twoja, twoje, twym, twój, ty, tych, tylko, tym, u, w, wam, wami, was, wasz, wasza, wasze, we, według, wiele, wielu, więc, więcej, wszyscy, wszystkich, wszystkie, wszystkim, wszystko, wtedy, wy, właśnie, z, za, zapewne, zawsze, ze, zł, znowu, znów, został, żaden, żadna, żadne, żadnych, że, żeby

Rysunek 2.4: Przykładowa lista słów przestankowych dla języka polskiego [40]

Ostatecznie każdy dokument zostaje więc odwzorowany przez wektor w przestrzeni dyskretnej, w której każdy wymiar odpowiada unikalnemu słowu. Poszczególne elementy wektora określać mogą:

- **Obecność termu w dokumencie**
  Każdy element wektora należy do zbioru {0;1}.

- **Liczbę wystąpień termu w dokumencie**
  Każdy element wektora należy do zbioru liczb naturalnych.

Przykład: Załóżmy, że zbiór uczący zawiera 3 dokumenty (pierwsza kolumna tabeli 2.2). W pierwszym kroku dokonujemy tokenizacji i lematyzacji dokumentów, dzięki czemu przekształcamy każdy z nich w multizbiór termów (druga kolumna tabeli 2.2). Przeprowadzenie lematyzacji na etapie *preprocessingu* nie jest konieczne i zależy wyłącznie od przyjętych założeń. Drugim krokiem jest stworzenie słownika, składającego

Dokument	**Termy**
Ala ma kota.	Ala, mieć, kot
Tomek ma psa, ale nie ma kota.	Tomek,mieć, pies, ale, nie, mieć, pies
Ala ma psa.	Ala, mieć, pies

Tabela 2.2: Przykładowy zestaw dokumentów

się ze zbioru wszystkich unikalnych termów. Dla przykładowego zbioru dokumentów otrzymujemy słownik zawierający następujące termy: Ala, ale, kot, mieć, nie, pies, Tomek. Termy w słowniku szeregujemy według pewnej stałej kolejności. Dysponując tak przygotowanym słownikiem, możemy wyrazić każdy dokument za pomocą wektora, w którym i-ty element odpowiada liczbie wystąpień i-tego termu w reprezentowanym dokumencie. Dla przykładowego zbioru dokumentów dostajemy wektorową reprezentację przedstawioną w tabeli 2.3.

		Wektor						
Lp.	**Dokument**	**Ala**	**ale**	**kot**	**mieć**	**nie**	**pies**	**Tomek**
1.	Ala ma kota.	1	0	1	1	0	0	0
2.	Tomek ma psa, ale nie ma kota.	0	1	1	2	1	1	1
3.	Ala ma psa.	1	0	0	1	0	1	0

Tabela 2.3: Wektorowa reprezentacja przykładowych dokumentów

2.2.2.2 Model n-gramowy

W podrozdziale 2.2.2.1 opisano podejście *bag-of-words*, będące najprostszym sposobem na przekształcenie dokumentu do ich reprezentacji wektorowej. Metoda ta bywa krytykowana z powodu nieuwzględniania zależności między pomiędzy termami [45]. Aby wykazać niedostatki BOW zauważmy tylko, że wektor nr 2 dwa tabeli 2.3. ([0 1 1 2 1 1 1]) może odpowiadać również zdaniu *Tomek ma kota, ale nie ma psa.* W pewnych zastosowaniach taka różnica może mieć znaczenie (np. gdybyśmy tworzyli klasyfikator określający, czy w dokumencie mowa jest o posiadaniu psa). W metodzie BOW termy są zasadniczo pojedynczymi wyrazami, chyba, że na etapie tokenizacji uznano pewne złożenie wyrazowe za pojedynczy token. Poza tymi wyjątkami, model BOW, używając dalej przykładu z tabel 2.2 i 2.3, nie odwierciedla różnicy między wystąpieniem obok siebie „nie ma psa" i „nie ma kota". Sposobem na uchwycenie tych relacji jest zastosowanie modelu n-gramowego. Polega on na wykorzystaniu jako termów ciągów o długości n tokenów. Kontynuując użycie przykładu, przyjmijmy model bigramowy, a więc taki, w którym pojedynczy term składa się z dwóch tokenów. Multizbiory termów dla przykładowego zestawu dokumentów przedstawiono w tabeli (przy zastosowaniu lematyzacji tokenów).

Dokument	**Temy**
Ala ma kota.	Ala mieć, mieć kot
Tomek ma psa, ale nie ma kota.	Tomek mieć, mieć pies, pies ale, ale nie, nie mieć, mieć kot
Ala ma psa.	Ala mieć, mieć pies.

Tabela 2.4: Przykładowe dokumenty i odpowiadające im multizbiory bigramów

Zbiorowi przykładowych dokumentów można wówczas przyporządkować wektory zaprezentowane w tabeli 2.5.

		Wektor						
Lp.	**Dokument**	**Ala mieć**	**mieć kot**	**Tomek mieć**	**mieć pies**	**pies ale**	**ale nie**	**nie mieć**
1.	Ala ma kota.	1	1	0	0	0	0	0
2.	Tomek ma psa, ale nie ma kota.	0	1	1	1	1	1	1
3.	Ala ma psa.	1	0	0	1	0	0	0

Tabela 2.5: Wektory przykładowych dokuemntów - model bigramowy

2.2.2.3 Częstość termów - odwrotna częstość dokumentów

Analizując teksty z języka naturalnego łatwo skonstatować, że nie wszystkie termy (zwykle: pojedyncze słowa) mają jednakowy wpływ na ogólny wydźwięk całego dokumentu. Powszechnie używaną metodą ważenia termów jest częstość termów - odwrotna częstość dokumentów (ang. *Term Frequency - Inverse Document Frequency,TFIDF*). Przyjmuje się w niej, że wpływ słowa na znaczenie dokumentu zależy od liczby wystąpień tego słowa w dokumentach należących do korpusu. TFIDF jest iloczynem częstości termów (TF) oraz odwróconej częstości dokumentów, zawierających dany term. Pierwszy czynnik TFIDF obliczmy według równania 2.2

$$TF_{i,j} = \frac{n_{i,j}}{|d_j|} \tag{2.2}$$

gdzie:
$n_{i,j}$ – liczba wystąpień i-tego termu w j-tym dokumencie d_j
$|d_j|$ - całkowita liczba wystąpień wszystkich termów w j-tym dokumencie d_j
Drugim czynnikiem równania TFIDF jest odwrotna częstość dokumentów, którą obliczamy według równania 2.3.

$$IDF_i = log\frac{|D|}{|\{d : x_i \in d\}|} \tag{2.3}$$

gdzie:
$|D|$ –liczba wszystkich dokumentów w zbiorze

$|\{d : x_i \in d\}|$ - liczba wszystkich dokumentów zawierających term x_i
Ważoną wartość i-tego elementu wektora obliczamy ostatecznie według równania 2.4.

$$TFIDF_{i,j} = TF_{i,j}I\dot{D}F_i \tag{2.4}$$

Używając przekształcenia na multizbiór termów z podejścia *bag of words*, uzyskujemy wektory termów ważone metodą TFIDF postaci, którą przedstawiono w tabeli 2.6. Możemy zauważyć na przykład, że występujący we wszystkich dokumentach term mieć otrzymuje wagę 0 (w praktyce zostaje więc uznany na nic nieznaczące słowo przestankowe).

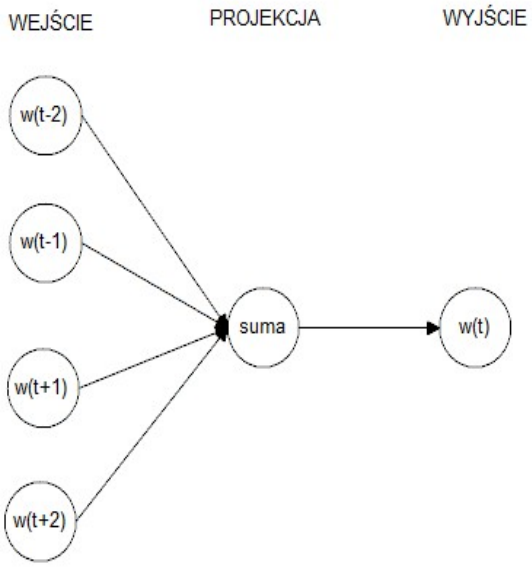
		Wektor						
Lp.	**Dokument**	**Ala**	**ale**	**kot**	**mieć**	**nie**	**pies**	**Tomek**
1.	Ala ma kota.	0.1351550	0	0.1351550	0	0	0	0
2.	Tomek ma psa, ale nie ma kota.	0	0.1569446	0.0579236	0	0.1569446	0.0579236	0.1569446
3.	Ala ma psa.	0.1351550	0	0	0	0	0.1351550	0

Tabela 2.6: Reprezentacja wektorowa z użyciem TFIDF

2.2.2.4 word2vec

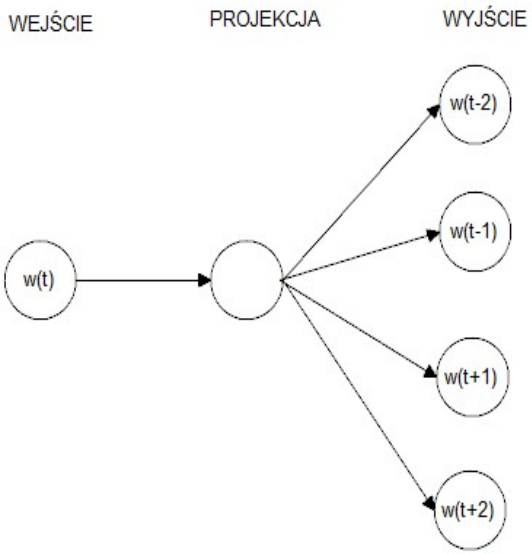
Innym sposobem na wychwycenie kontekstu, w jakim pojawia się dany term (dane słowo), jest wektorowa reprezentacja słowa. Wśród najpopularniejszych metod transformacji zbioru dokumentów na zbiór wektorów reprezentujących termy występujące w tym zbiorze wymienić można word2vec. Nie jest to nazwa jednego algorytmu, lecz ich grupy, spośród których należy wspomnieć przede wszystkim o dwóch:

- **Continuous Bag of Words (CBOW)**
  W modelu CBOW staramy się przewidzieć kolejne słowo w zależności od kontekstu.



Rysunek 2.5: Model CBOW [23]

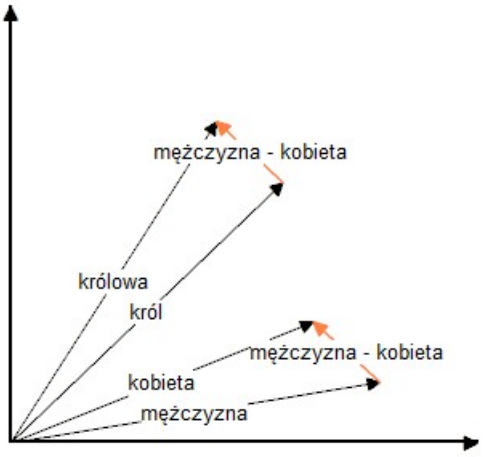
- **Skip-gram**
  W modelu skip-gram staramy się przewidzieć kontekst w zależności od słowa.



Rysunek 2.6: Model Skip-gram

Każdy z węzłów warstwy wejścia jest wektorem tej samej długości, natomiast zapis

$w(t)$ oznacza słowo znajdujące się na t-tym miejscu w zdaniu (dokumencie) licząc od jego początku. Niniejsza praca nie dotyczy szczegółowej analizy algorytmów z grupy word2vec. Wystarczy więc omówić wynik uzyskiwane przez zastosowanie implementacji algorytmów z rodziny word2vec. Po wytrenowaniu modelu na dużym korpusie dostajemy reprezentację wektorową, która lokuje wyrazy bliskoznaczne niedaleko siebie oraz zachowuje położenie między wektorami stosowne do ich semantycznej zależności. Jeśli przyjąć dla uproszczenia, że wektory słów umiejscawiane są w przestrzeni dwuwymiarowej, powinniśmy np. dostać wektor dla słowa królowa poprzez wykonanie operacji król + (mężczyzna – kobieta). Zostało to przedstawione na rysunku 7. zaczerpniętym z opracowania na temat wektorowego modelu słownika języka polskiego wytrenowanego na tekstach polskich pochodzących z korpusu Common Crawl [34].



Rysunek 2.7: word2vec dla języka polskiego [34]

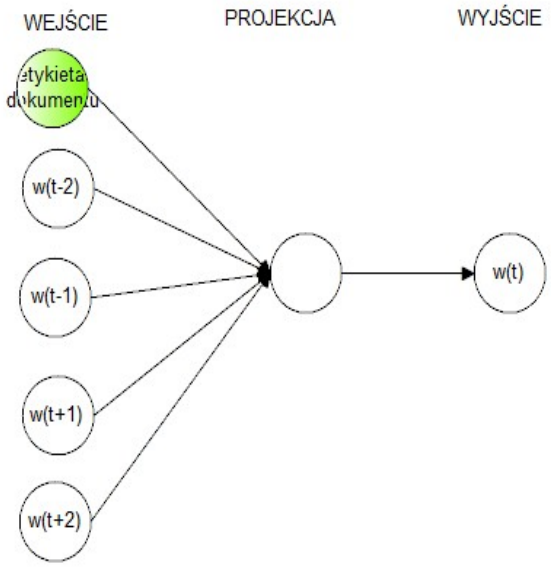
Osobną kwestią jest wykorzystanie wytrenowanego modelu w kolejnej fazie uczenia maszynowego. Możliwe jest np. przypisanie dokumentowi wektora będącego średnią z wektorów termów występujących w tym dokumencie, jak zaproponowano to w tutorialu dostępnym na portalu Kaggle [15].

2.2.2.5 doc2vec

Doc2vec (lub paragraph2vec) to zbiorcza nazwa odnosząca się zasadniczo do grupy składającej się z dwóch algorytmów:

- Distributed memory model
- Distributed bag of words

Tym co odróżnia algorytmy z grupy *doc2vec* od algorytmów z grupy *word2vec* to możliwość osadzenia w przestrzeni wektorowej nie tylko pojedynczych słów, ale również całych dokumentów. Modyfikacją, która umożliwia takie przekształcenie jest dodanie na wejściu dodatkowego parametru – wektora przyporządkowanego do etykiety konkretnego dokumentu. Przykładowo, schemat modelu *distributed memory* będącego rozszerzeniem modelu CBOW znanego z word2vec przedstawia się, jak pokazano na rysunku 2.8. Dzięki uzyskanemu modelowi jesteśmy w stanie określić dystans nie tylko pomiędzy



Rysunek 2.8: Distributed memory model

poszczególnymi termami, ale także między dokumentami.

2.3 Transformacja wektorów danych

W podrozdziale 2.1.1 przyjęto zapis $\langle \mathbf{x}, \mathbf{y} \rangle$ dla oznaczenia zbioru uczącego, przy czym jako x oznaczono zbiór wektorów stanowiących reprezentacje przykładów. Dla odróżnienia przyjmijmy więc, że zbiór wektorów poddanych transformacjom oznaczymy jako $\mathbf{x}'$. Transformacje wektorów danych są rodzajem funkcji, która przekształca wejściowy zbiór wektorów $\mathbf{x}$ na zbiór wynikowy $\mathbf{x}'$. Wektory z nowego zbioru $\mathbf{x}'$ mogą różnić się liczbą wymiarów od wektorów ze zboiru wejściowego $\mathbf{x}$. Typowe rodzaje transformacji, jakim można poddać wektor wejściowy to [12] m.in.:

- **Standaryzacja**
  Standaryzacja stosowana jest, gdy cechy opisują te same własności w inny sposób. Przykładowo w $\mathbf{x}$ cecha *długość* wyrażona jest zarówno w metrach i centymetrach. Przyjmujemy jednolity sposób reprezentacji długości np. w centymetrach.

- **Normalizacja**
  Załóżmy, że wektory ze zbioru $\mathbf{x}$ opisują obrazy, a jedną z zawartych w nich cech jest liczba czerwonych pikseli. Aby móc porównywać obrazy różnej wielkości, cechę tę wyrażamy nie za pomocą wielkości bezwzględnej, ale stosunku liczby pikseli czerwonych to liczby wszystkich pikseli składających się na dany obraz.

- **Zmniejszanie liczby wymiarów (cech)**
  Tę transformację stosuje się w przeprowadzeniu analizy głównych składowych (ang. *Principal Component Analysis, PCA*). W jej wyniku zastępujemy dwie zmienne (cechy, wymiary) jedną zmienną składową. Przykładowo, załóżmy, że prognozujemy, ile bramek zdobędzie w danym sezonie napastnik, a wśród opisującego go wektora mamy takie cechy jak liczba bramek zdobytych w poprzednim sezonie oraz liczba strzałów celnych w poprzednim sezonie. Jeżeli te dwie wartości okażą się ze sobą silnie skorelowane, możemy je zastąpić jedną zmienną skuteczność w poprzednim sezonie.

- **Zwiększanie liczby wymiarów cech (cech)**
  Tę transformację stosuje się np. poprzez dodanie cech, których wartości są iloczynami cech istniejących na początku.

- **Dyskretyzacja**
  Jest to Zamiana zamiana wartości z przestrzeni ciągłej na wartości z przestrzeni dyskretnej. Przykładowo, możemy zamienić cechę cena na cechę klasa cenowa – pierwotna wartość zostaje zaliczona do pewnego przedziału.

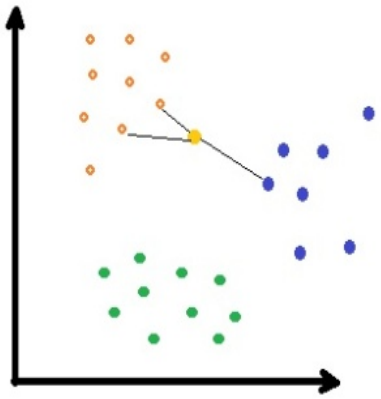
2.4 Nadzorowane metody automatycznej klasyfikacji

2.4.1 K-najbliższych sąsiadów

Metodę k-najbliższych sąsiadów (ang. k-nearest neighbors, k-NN) możemy określić mianem metody opartej na pamięci[6] . Etap trenowania systemu polega bowiem, w najprostszej wersji k-NN, wyłącznie na zapamiętaniu całego zbioru . Klasyfikacja nowego przykładu polega na przypisaniu mu tej kategorii, do której należy najwięcej spośród k najbliżej leżących przykładów ze zbioru . Dla k=1 algorytm ten można sformalizować za pomocą równania 2.5

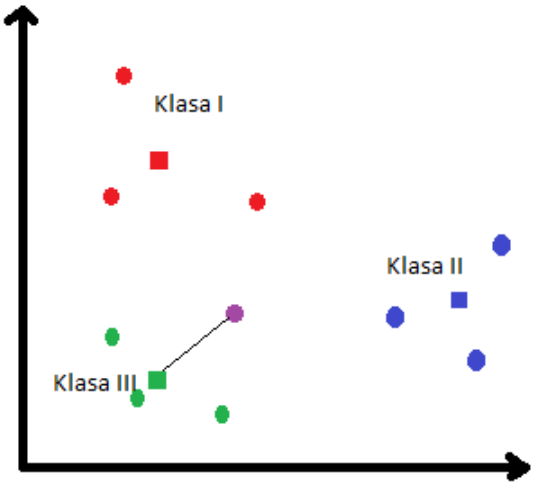
$$h(x) = f(argmin_{x_i \in \mathbf{x}} \delta(x_i, \overline{x})) \tag{2.5}$$

gdzie $\delta(x_i, \overline{x})$ oznacza odległość euklidesową między punktami wyznaczanymi przez wektory x_i oraz $\overline{x}$. Na rysunku 2.9. przedstawiono przykład działania algorytmu dla k=3. Nowy przykład nieuwzględniony w procesie uczenia (żółty punkt), zostaje zaklasyfikowany na podstawie odległości od punktów reprezentujących przykłady ze zbioru uczącego. Jak widzimy, wśród 3 najbliższych sąsiadów nowego przykładu znalazły się 2 przykłady należące do klasy pomarańczowej i jeden należący do klasy niebieskiej. Wynikiem działania algorytmu k-NN jest zaklasyfikowanie nowego przykładu do klasy pomarańczowej.



Rysunek 2.9: Klasyfikacja 3-NN

Modyfikacją metody kNN jest algorytm najbliższy centroid (ang. *Nearest Centroid*). Etap uczenia polega na wyznaczeniu centroidów dla poszczególnych klas, czyli środków przestrzeni zajmowanej przez punkty należące do danej klasy. W celu dokonania klasyfikacji sprawdza się odległość między nowym przykładem, a centroidem każdej z klas. Nowy przykład zostaje zaliczony do tej klasy, od której centroidu znajduje się najbliżej. Zostało to zilustrowane na rysunku 2.10. – nowy przykład przedstawiono za pomocą fioletowego punktu, a centroidy klas jako kwadraty o odpowiednich kolorach. Nowy przykład zostaje zaliczony do klasy III.



Rysunek 2.10: Algorytm najbliższy centroid

2.4.2 Naiwny klasyfikator Bayesa

Jednym z najczęściej stosowanych metod uczenia maszynowego jest klasyfikator oparty na twierdzeniu Bayesa. W tym miejscu warto podkreślić, że w zależności od charakterystyki tego wektora występują różne wersje naiwnego klasyfikatora Bayesa (ang. *Naive Bayes Classifier, NBC*). Bywają one ze sobą mylone. Są one oparte na następujących rozkładach prawdopodobieństwa:

- **Rozkład Gaussa**
  Wektor cech zawiera wartości z przestrzeni ciągłej. (W przypadku przetwarzania języka rozpatrywać będziemy tylko wektory cech w przestrzeniach dyskretnych.)

- **Rozkład zero-jedynkowy (ang. *Bernoulli distribution*)**
  Wektor cech zawiera jedynie informacje o występowaniu bądź niewystępowaniu danej cechy.

- **Rozkład wielomianowy**
  Wektor cech zawiera informację o liczebności poszczególnych termów w multizbiorze.

W przypadku klasyfikacji tekstów udowodniono, że większą dokładność klasyfikacji zapewnia naiwny klasyfikator Bayesa oparty o rozkład wielomianowy (w porównaniu z rozkładem zero-jedynkowym)[21]). Niezależnie od przyjętego rozkładu prawdopodobieństwa, wynik klasyfikacji otrzymujemy poprzez zastosowanie następującego algorytmu:

1. Obliczenie prawdopodobieństwa wystąpienia słowa x_j w klasie c_i

$$P(x_j|c_i) = \frac{wystąpienia\ termu\ x_j\ w\ dokumentach\ klasy\ c_i}{wystąpienia\ wszystkich\ termów\ w\ dokumentach\ klasy\ c_i} \tag{2.6}$$

2. Obliczanie prawdopodobieństwa przynależności do danej klasy pod warunkiem wystąpienia danego wektora cech.

$$P(c_i|d') = P(c_i|x_1, ..., x_j) = \frac{P(c_i) \cdot P(x_1, ..., x_j|c_i)}{P(x_1, ..., x_j)} \tag{2.7}$$

3. Wybór klasy, dla której obliczone prawdopodobieństwo jest największe.

$$\overline{c} = argmax_{c_i} P(c_i|x_1, ..., x_j) \tag{2.8}$$

Dokument	Termy	Klasa
d_1	trener, piłka, rzut	Piłka nożna
d_2	trener, parkiet, zespół, mecz, rzut, rzut, rzut, piłka	Koszykówka
d_3	zespół, mecz, boisko, zawodnik, zespół	Piłka nożna
d_4	zawodnik, zawodnik, wynik, piłka	Piłka nożna
d'	rzut, zawodnik, trener	?

Tabela 2.7: Przykładowy zbiór dokumentów

Przedstawmy obliczanie NBC na przykładzie zbioru dokumentów zaprezentowanego w tabeli 2.7. Dla dokumentu klasyfikowanego na podstawie zbioru d_1, d_2, d_3, d_4 otrzymujemy zgodnie z równaniami 2.6 i 2.7:

$$P(pn|rzut, zawodnik, trener) = \frac{\frac{3}{4} \cdot \frac{1}{12} \cdot \frac{1}{4} \cdot \frac{1}{6}}{\frac{3}{20} \cdot \frac{1}{10} \cdot \frac{1}{10}} \approx 0,347 \tag{2.9}$$

$$P(k|rzut, zawodnik, trener) = \frac{\frac{1}{4} \cdot \frac{3}{8} \cdot \frac{1}{8} \cdot \frac{1}{8}}{\frac{3}{20} \cdot \frac{1}{10} \cdot \frac{1}{10}} \approx 0,098 \tag{2.10}$$

Zgodnie z równaniem 2.8 zaliczamy dokument d' do klasy piłka nożna (pn).

2.4.3 Komplementarny naiwny klasyfikator Bayesa

Komplementarny naiwny klasyfikator Bayesa (ang. *Complement Naive Bayes, CNB*) to klasyfikator wprowadzony w celu zmniejszenia niepożądanego wpływu różnej liczebności klas w zbiorze uczącym [29]. Autorzy tego zaprezentowanego w 2003 r. algorytmu twierdzą, iż klasyczna wersja NBC przejawia tendencję do tzw. biasu, nadmiernie "faworyzując" klasy najliczniej reprezentowane w zbiorze uczącym. Klasyfikacja nowego przypadku na podstawie zbioru uczącego przebiega następująco:

1. Obliczanie komplementarnego prawdopodobieństwa przynależności do danej klasy. Prawdopodobieństwo to stanowi odwrotność prawdopodobieństwa, że dany przypadek należy do wszystkich innych klas niż klasa aktualnie rozpatrywana.

$$P_{CNB}(c_i|x_1, ..., x_j) = P(\tilde{c}_i) \cdot \frac{1}{P(x_1, ..., x_j|\tilde{c}_i)} \tag{2.11}$$

gdzie:

$\tilde{c}_i$ - zbiór zawierający wszystkie przypadki nienależące do klasy c_i

2. Wybór klasy, dla której obliczone prawdopodobieństwo jest najmniejsze.

$$\overline{c} = argmin_{c_i} P(c_i|x_1, ..., x_j) \tag{2.12}$$

W równaniach 2.13 oraz 2.14 przedstawiono obliczanie prawdopodobieństwa komplementarnego zgodnie z równaniem 2.11. Zauważmy, że prawdopodobieństwo komplementarne, w przeciwieństwie do prawdopodobieństwa klasycznego nie należy do przedziału $\langle 0; 1\rangle$

$$P(pn|rzut, zawodnik, trener) = \frac{\frac{1}{4}}{\frac{3}{8} \cdot \frac{1}{8} \cdot \frac{1}{8}} \approx 42,373 \tag{2.13}$$

$$P(k|rzut, zawodnik, trener) = \frac{\frac{3}{4}}{\frac{1}{12} \cdot \frac{1}{4} \cdot \frac{1}{6}} \approx 214,286 \tag{2.14}$$

Zgodnie z regułą z równania document *d'* zaliczamy do klasy *piłka nożna*.

2.4.4 Negacyjny Naiwny Klasyfikator Bayesa

Negacyjny naiwny klasyfikator Bayesa (ang. *Negation Naive Bayes, NNB*) został opracowany w celu osiągnięcia tego samego w tym samym celu, co algorytm CNB. Jego autorzy twierdzą jednak, iż komplementarny naiwny klasyfikator Bayesa nie wynika z równania używanego w klasycznym naiwnym klasyfikatorze Bayesa [16]. Ponadto w przytoczonej pracy zarzuca się, że CNB w żaden sposób nie zmienia sposobu obliczania prawdopodobieństwa wystąpienia i-tej klasy ($P(c_i)$), co również powinno zostać uwzględnione przy dużej skośności (asymetrii) rozkładu prawdopodobieństwa. Regułę decyzyjną NNB uzyskujemy zamieniając w równaniu 2 sposób wyliczania prawdopodobieństwa warunkowego (równanie 2.15).

$$\overline{c} = argmax_{c_i} \frac{1}{1 - P(c_i} \cdot \frac{1}{P(x_1, ..., x_j|c_i)} \tag{2.15}$$

Korzystając z równania 2.15, obliczamy „prawdopodobieństwo" NNB. Podobnie jak w przypadku CNB wartości nie muszą się zawierać w przedziale $\langle 0; 1\rangle$. Obliczenia przedstawiono w równaniach 2.16 i 2.17.

$$P_{NNB}(pn|rzut, zawodnik, trener) = \frac{1}{\frac{3}{4}} \cdot \frac{1}{\frac{1}{12} \cdot \frac{1}{4} \cdot \frac{1}{6}} \approx 380,857 \tag{2.16}$$

$$P_{NNB}(k|rzut, zawodnik, trener) = \frac{1}{\frac{1}{4}} \cdot \frac{1}{\frac{3}{8} \cdot \frac{1}{8} \cdot \frac{1}{8}} \approx 677,966 \tag{2.17}$$

Zgodnie z regułą z równania 2.15 dokument *d'* zaliczamy do klasy koszykówka (k).

2.4.5 Maszyna wektorów nośnych

Algorytm nazywany maszyną wektorów nośnych (ang. *Support Vector Machine, SVM*) opiera się w dużej mierze na rzutowaniu danej przestrzeni cech na przestrzeń o większej liczbie wymiarów. Zwiększenie liczby wymiarów pozwala osiągnąć własność zwaną liniową separowalnością, tj. możliwość rozdzielenia podzbiorów $\langle \mathbf{x}, \mathbf{y} \rangle_k \in \langle \mathbf{x}, \mathbf{y} \rangle$ (gdzie k oznacza k-tą klasę) za pomocą hiperpłaszczyzny wyznaczanej z równania 2.18.

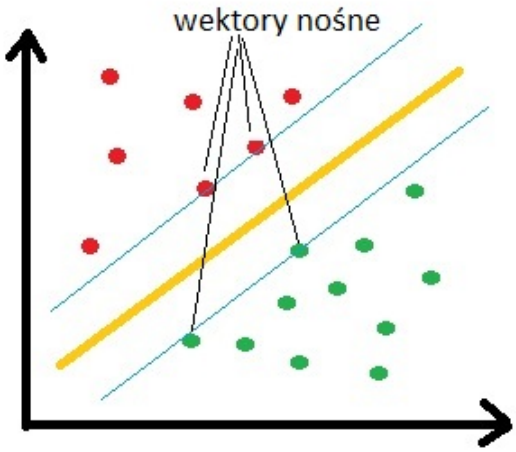
$$H(\mathbf{w}, \mathbf{b}) = \mathbf{x}|\mathbf{w}^T + \mathbf{b} = 0 \tag{2.18}$$

Funkcja decyzyjna przyjmuje wartości dodatnie, jeżeli przykład znajduje się nad hiperpłaszczyzną i ujemne, jeżeli leży pod nią. Należy przy tym zauważyć, że najprostsza wersja SVM jest klasyfikatorem binarnym – przestrzeń cech dzielona jest na dwie części zawierające przykłady należące do dwóch różnych klas. Aby użyć SVM do klasyfikacji wieloklasowej możemy zastosować następujące podejścia [13]:

- **One-againts-all**
  Konstruujemy n binarnych modeli SVM, przy czym w każdym do trenowania używamy dwóch podzbiorów $\langle \mathbf{x}, \mathbf{y} \rangle_k$ oraz $\langle \mathbf{x}, \mathbf{y} \rangle_{\text{nie należące do klasy k}}$ (nie należące do klasy k) (gdzie n oznacza liczbę klas). Otrzymujemy tym samym n funkcji decyzyjnych. Przykładowi $\overline{x}$ przypisujemy tę klasę, dla której wartość funkcji decyzyjnej jest największa.

- **One-againts-one**
  Konstruujemy $\frac{n \cdot (n-1)}{2}$ modeli SVM wykorzystując wszystkie możliwe pary $\langle \mathbf{x}, \mathbf{y} \rangle_k$, $\langle \mathbf{x}, \mathbf{y} \rangle_j \in \langle \mathbf{x}, \mathbf{y} \rangle$ przy czym $k \neq j$. Otrzymujemy $\frac{n \cdot (n-1)}{2}$ klasyfikatorów – ostateczną decyzję możemy podjąć np. poprzez głosowanie (która klasa jest wskazywana najczęściej).

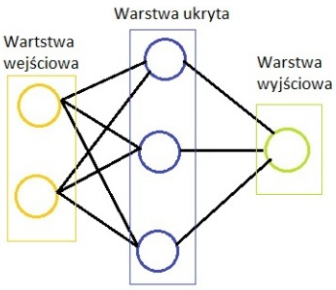
- **Directed Acyclic Graph Support Vector Machines (DAGSVM)**
  Etap trenowania taki sam jak w One-against-one. Klasyfikatory tworzą wewnętrzne węzły skierowanego grafu acyklicznego, którego liśćmi są klasy.



Rysunek 2.11: Maszyna wektorów nośnych - schemat działania algorytmu

2.4.6 Sieci neuronowe

Sztuczne sieci neuronowe (lub po prostu sieci neuronowe, ang. *artificial neural networks, ANN, NN*) to grupa algorytmów uczenia maszynowego powstała jako próba zasymulowania procesów zachodzących w mózgach organizmów żywych ze względu na ich interesujące właściwości [17]. Same sieci neuronowe w różnych wariantach tworzą niezwykle złożoną dziedzinę uczenia maszynowego zwaną głębokim uczeniem maszynowym (ang. *deep learning*). W tym podrozdziale przedstawiona zostanie zasada funkcjonowania ich najprostszej wersji tj. sieci jednokierunkowe (ang. *feedforward*).



Rysunek 2.12: Schemat prostej sieci neuronowej typu feedforward

Na rysunku 2.12. widzimy schemat prostej sieci neuronowej typu *feedforward*. Sieci te składają się z 3 warstw:

- **Wejściowej**. Zawiera tyle neuronów, ile cech znajduje się w danym modelu (ile wymiarów mają wektory wejściowe)
- **Ukrytej**. Sama warstwa ukryta może zawierać wiele warstw neuronów, przy czym ostatnia warstwa musi zawierać tyle neuronów, ile występuje możliwych klas w danym zadaniu klasyfikacji.
- **Wyjściowej**. W przypadku problemu klasyfikacji zawiera dokładnie jeden neuron.

Jak widzimy na rysunku 2.12, każdy neuron z jednej warstwy jest połączony z każdym neuronem z warstwy następnej. W procesie klasyfikacji zachodzi jednokierunkowy przepływ danych (w przypadku sieci z rysunku 2.12 – od lewej do prawej). Każdej krawędzi łączącej neurony przyporządkowana jest pewna waga, przez którą mnożona jest wartość „przekazywana" po tej krawędzi. Gdy wartość liczbowa dociera do neuronu, zostaje podstawiona do pewnej funkcji aktywacji (zwykle funkcji sigmoidalnej) takiej jak np.:

- Funkcja logistyczna
$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.19}$$
- Funkcja tangens hiperboliczny
$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.20}$$
- Funkcja rectifier (rectified linear unit, ReLU)
$$f(x) = max(0, x) \tag{2.21}$$

W ostatnim, wyjściowym neuronie wykorzystywana jest funkcja softmax, która zamienia wektor uzyskany z poprzedniej warstwy na wektor prawdopodobieństw przynależności do poszczególnych klas. Funkcję softmax opisuje się równaniem 2.22.

$$\omega(\mathbf{z})_j = \frac{\mathbf{e}^{z_j}}{\sum_{n=1}^{n} \mathbf{e}^{z_i}} \tag{2.22}$$

Według tego wzoru wyznaczamy j-ty element wektora wyjściowego. Ostateczną decyzję podejmujemy przyporządkowując nowemu przykładowi $\overline{x}$ tę klasę, której w wyjściowym wektorze odpowiada najwyższe prawdopodobieństwo.

2.5 Ewaluacja modelu

2.5.1 Walidacja na odłożonym zbiorze testowym

Walidacja na odłożonym zbiorze testowym (ang. *hold-out validation*) polega na jednokrotnym podziale zbioru przykładów na dwie części: zbiór uczący i zbiór testowy. Należy zadbać o to, aby rozkład przykładów należących do poszczególnych klas był zbliżony w obu zbiorach.

2.5.2 K-krotna walidacja krzyżowa

K-krotna walidacja krzyżowa (ang. k-fold cross-validation) przebiega w następujący sposób [6]:

- Dzielimy losowo zbiór uczący na k równolicznych podzbiorów
- Trenujemy i walidujemy system k razy, przy czym w każdym przypadku mamy:
  - Zbiór uczący: jeden z k podzbiorów
  - Zbiór trenujący: suma pozostałych k-1 podzbiorów
- Uśredniamy uzyskane wyniki walidacji

Każdy przykład ze zbioru danych zostaje użyty k-1 razy w procesie trenowania oraz dokładnie jeden raz w fazie testowania (walidacji).

2.5.3 Miary jakości

2.5.3.1 Dokładność

Dokładność (ang. *accuracy*) jest procentem, jaki wśród wszystkich odpowiedzi klasyfikatora stanowią odpowiedzi prawidłowe. Za odpowiedź prawidłową uznajemy etykietę przypisaną przykładowi należącemu do zbioru uczącego. Podczas trenowania klasyfikatorów za wersję baseline można przyjąć naiwny klasyfikator przypisujący każdemu przykładowi ze zbioru testowego etykietę najliczniej reprezentowanej klasy. Zakładając, że znamy procentowy rozkład klas w zbiorze testowym (np. jest on taki sam jak w zbiorze uczącym), jesteśmy w stanie uzyskać klasyfikator w wersji baseline o dokładności równej odsetkowi najliczniejszej klasy.

	Wynik pozytywny	**Wynik ngatywny**
Naprawdę pozytywne	Prawidłowy wynik pozytywny (ang. true positive, TP)	Błędny wynik pozytywny (ang. false positive, FP)
Naprawdę negatywne	Prawidłowy wynik negatywny (ang. true negative, TN)	Błędny wynik negatywny (ang. false negative, FN)

Tabela 2.8: Możliwe wyniki klasyfikacji binarnej

2.5.3.2 Precision i recall

Precision i recall są pojęciami dotyczącymi klasyfikacji binarnej. Zakładamy przy tym, że chodzi nam o wykrycie pewnego zjawiska, stąd też jedna z klas odpowiada występowaniu tego zjawiska (wynik pozytywny), druga zaś jego brakowi (wynik negatywny). W związku z tym wyniki klasyfikacji przeprowadzonej na zbiorze testowym przynależą do jednej z czterech grup w zależności od klasy zaproponowanej przez klasyfikator oraz tego, czy owa klasa została przypisana prawidłowo czy też błędnie. Grupy te zostały przedstawione w tablicy pomyłek (ang. *confusion matrix*) - tabela 2.8. Dzięki oznaczeniom opisanym w tabeli pomyłek, możemy przedstawić wzory na dwie miary jakości klasyfikacji wymienione w tytule podrozdziału: precision i recall.

- **Precision**
  Precision jest określa skuteczność w wykrywaniu zjawiska tj. stosunek prawidłowych wyników pozytywnych do wszystkich liczby wszystkich przykładów zaklasyfikowanych jako pozytywne (równanie 19).

$$\mathbf{precision} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP}} \tag{2.23}$$

- **Recall**
  Recall jest stosunkiem prawidłowych wyników pozytywnych do sumy prawidłowych wyników pozytywnych oraz błędnych wyników negatywnych

$$\mathbf{recall} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}} \tag{2.24}$$

Interpretacja wyników miar *precision* i *recall* zależy przede wszystkim od rozpatrywanego problemu klasyfikacji. Wysoki wynik miary precision oznacza, że wśród przykładów

określonych przez klasyfikator jako pozytywne, wysoki ich odsetek został zaklasyfikowany prawidłowo. Przykładowo, jeśli korzystamy z klasyfikatora spamu o precision równym 95% możemy stwierdzić, że wśród odrzuconego spamu 5% stanowią wiadomości treściwe (nie-spam). Wysoki wynik recall świadczy o tym o procencie przypadków pozytywnych, które klasyfikator jest w stanie prawidłowo zidentyfikować. Łatwo jest osiągnąć wysoki wynik recall stosując naiwny klasyfikator działający wg reguły „zawsze rezultat pozytywny". Wysoka wartość tej miary może być pożądana np. w dla klasyfikatora rozpoznającego nowotwory (wolimy, aby klasyfikator wykrył wszystkie przypadki występowania choroby nawet kosztem fałszywych alarmów).

ROZDZIAŁ 3

Gromadzenie danych do uczenia maszynowego

3.1 Pozyskiwanie danych z Internetu

Wydajne pozyskiwanie dużych ilości danych tekstowych z Internetu jest zasadniczo możliwe do zrealizowania na dwa sposoby:

- **Wykorzystanie programistycznego interfesju aplikacji (ang. *Application Programming Interface, API*)**
  Modelowym przykładem jest np. API Twittera, które umożliwia łatwe pozyskiwanie tweetów np. według miejsca, z którego zostały wysłane.

- **Wykorzystanie crawlerów**
  Można również automatycznie ekstrahować treści ze stron internetowych, co jest możliwe dzięki ustrukturyzowaniu stron WWW przez znaczniki HTML. Rysunek 3.1 przedstawia fragment źródła pochodzący z portalu Wirtulna Polska zawierający jeden komentarz pod artykułem.

Rysunek 3.1: Fragment źródła jednej ze stron portalu Wirtualna Polska [1]

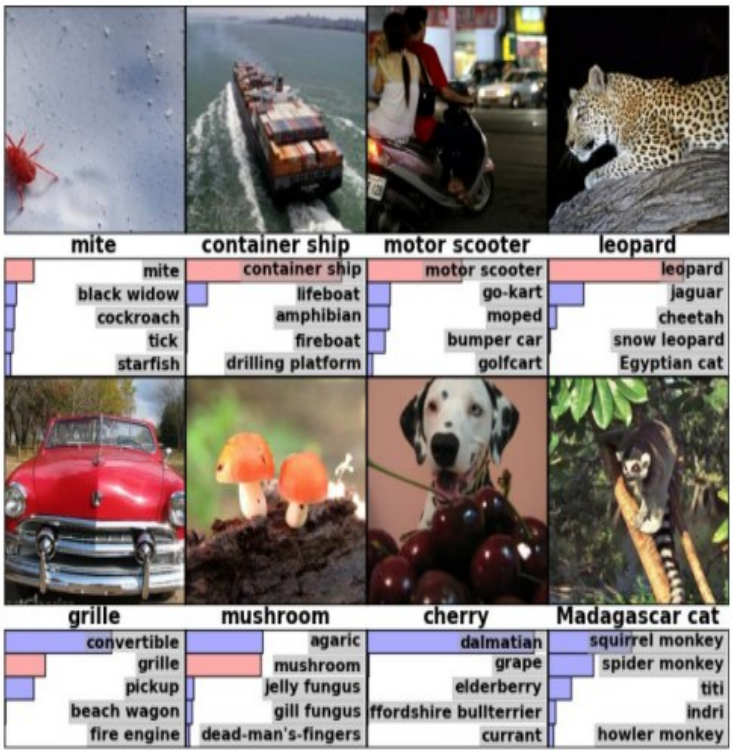
[1] http://wiadomosci.wp.pl/kat,1356,title,Premier-Belgii-ostrzega-ze-w-Europie-dojdzie-do-nowych-zamachow,wid,18247923,wiadomosc.html, dostęp: 2.06.2016

3.2 Etykietowanie danych przez ludzi

W przypadku uczenia nadzorowanego potrzebny jest zbiór danych oznaczonych etykietami. Etykiety te muszą zostać przypisane przez ludzi, gdyż dopiero na tej podstawie można wytrenować automatyczny klasyfikator. Ten sposób pozyskiwania danych, oprócz pracochłonności, często pociąga za sobą następujące wyzwania:

- Błędne etykietowanie
- Subiektywizm
- Zmienność ocen w czasie

Nawet jedna osoba może w różnych momentach oznaczyć dokładnie te same dane w odmienny sposób. Ilustrują to chociażby badania opisane w pracy [20] , w której porównano zgodność etykiet, jakie zostały nadane przez 15 osób temu samemu zbiorowi danych (zbiorowi stron internetowych) w odstępie czterech tygodni. Po powtórnym oznaczeniu danych nowe etykiety pokrywały się z etykietami pierwotnymi średnio w 81,7% . Wpływ subiektywnego spojrzenia na etykietowanie danych możemy łatwo zauważyć na przykładzie badań z dziedziny automatycznego rozpoznawania obrazów. Rysunek 3.2 przedstawia skuteczność klasyfikatora obrazów opisanego w pracy dotyczącej trenowania konwolucyjnych sieci neuronowych na zasobach sieci ImageNet [19]. Podpis pod każdym z obrazków oznacza klasę prawidłową (etykietę nadaną przez człowieka), natomiast poziome słupki poniżej obrazków oznaczają prawdopodobieństwa klas zwrócone przez klasyfikator. Środkowy obrazek w dolnym rzędzie na rysunku 3.2 przedstawia błąd klasyfikatora, który przypisał temu przykładowi etykietę *dalmatian* zamiast *cherry*. Należy tutaj jednak zauważyć, że samo oznaczenie rzeczonego obrazka właśnie etykietą *cherry* można uznać za dyskusyjne i obrazuje to wpływ subiektywnej oceny człowieka na charakterystykę zbioru uczącego. Sam proces manualnego oznaczania danych może zostać przeprowadzony w różny sposób. Skoro np. w przypadku gromadzenia danych do treningu automatycznego filtra spamu „wąskim gardłem jest przeciętna cierpliwość użytkownika w oznaczaniu dużej liczby e-mali" [46] rozwiązaniem rozwiązaniem może okazać się:

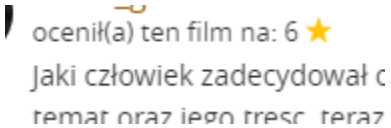


Rysunek 3.2: Wyniki automatycznej klasyfikacji obrazu [19]

- **Crowdsourcing** – na przykład tzw. „sztuczna sztuczna inteligencja” (ang. *artificial artificial intelligence*) w rodzaju Amazon Mechanical Turk. Przykładem jest mechanizm reCAPTCHA (rysunek 3.3), w którym użytkownik musi wpisać dwa słowa przedstawione na dwóch obrazkach. Jedno jest słowem kontrolnym („testem na człowieczeństwo”), drugie zaś, po pozytywnym przejściu testu zostaje użyte jako etykieta drugiego obrazka. Dzięki temu mechanizm dostarcza danych do treningu narzędzi OCR [30].

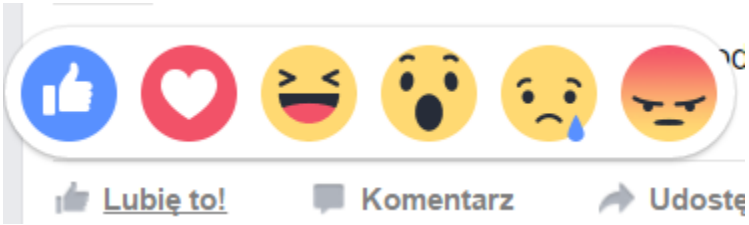
Rysunek 3.3: Mechanizm reCAPTCHA

- **Wykorzystanie ustrukturyzowanych lub oznaczonych treści**
  Warto zauważyć, że oprócz projektów takich jak Open Directory Project (ustrukturyzowany katalog stron internetowych), wszystkie strony internetowe posiadają pewną strukturę ułatwiającą ich przeglądanie (np. podział na kategorie artykułów: wiadomości, finanse, sport itd.). Ponadto na wielu portalach internetowych użytkownicy mają możliwość zamieszczania własnych ocen różnego rodzaju treści np. filmów (patrz rysunek 3.4), komentarzy (rysunek 3.5) Dotyczy zwłaszcza portali społecznościowych – za dobry przykład może posłużyć system wprowadzony na portalu Facebook w lutym 2016 r. (rysunek 3.6).



Rysunek 3.4: Ocenianie filmów na portalu Filmweb

Rysunek 3.5: System oceniania komentarzy na portalu WIrtualna Polska



Rysunek 3.6: System oceny treści na Facebooku

ROZDZIAŁ 4

Automatyczne filtrowanie i klasyfikacja treści

4.1 Filtrowanie spamu

Jedną z najstarszych technik używanych do filtrowania e-maili jest naiwny klasyfikator Bayesa. Za pierwszą implementację (1996) narzędzia tej klasy uważa się program ifile autorstwa Jasona Rennie służący do automatycznego sortowania e-maili [25]. Odpowiedzią spamerów było pojawienie się zespołu technik znanych jako *Bayes poisoning* polegających na odpowiednim manipulowaniu treścią w celu oszukania klasyfikatora. Do spamu dodawane są:

- Losowe słowa [42]
- Popularne słowa [33]
- Frazy typowe dla maili treściwych (ang. Ham e-mail) [32]

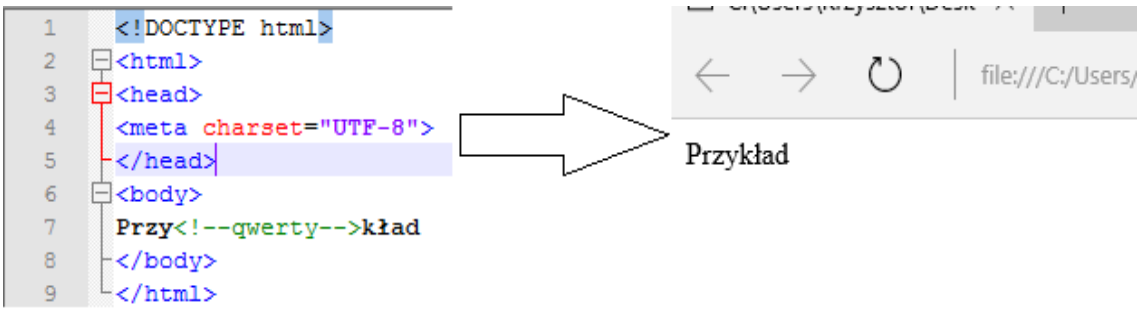
Wśród pozostałych czynników wpływających negatywnie na poziom skuteczności klasyfikatorów (technik spamerskich) wymienić można:

- **Niekonwecjonalny zapis**
  Niekonwencjonalny zapis może zostać użyty w celu ukrycia słów stanowiących o spamowym charakterze danego e-maila. Słowo jest zapisywane tak, aby jednocześnie było zrozumiałe dla człowieka i nietrywialne w interpretacji przez maszynę. Przykładowo, kojarzone ze spamem słowo Viagra może być zapisane jako Vviagra,

Via.gra, V1agra lub V—agra. Błędna tokenizacja, jak np. rozbicie słowa Via.gra na dwa tokeny „via” i „gra” niekoniecznie musi wpłynąć na pogorszenie skuteczności filtra. Jeżeli przytoczone tutaj dwa błędne tokeny częściej będą pojawiać się w klasie „spam”, nie zaburzą one ostatecznego wyniku [35]

- **Zaciemnianie**

  Przy zaciemnianiu tekstu mogą zostać użyte np. własności języka znaczników HTML. Słowo może wyświetlać się w sposób czytelny dla człowieka, będąc jednocześnie zaciemnione na poziomie surowego tekstu. Podobne techniki znajdują również zastosowanie, które można określić jako pozytywne, gdyż są wykorzystywane do w obronie przed botami zbierającymi adresy e-mail. Przykład zaciemniania pokazano na rysunku 4.1: tekst widoczny w przeglądarce jako *Przykład*, w pliku HTML przedzielony jest znacznikiem komentarza. Automatyczne (tj. za pomocą bota) odczytanie tego tekstu wymaga odpowiedniego parsowania.



Rysunek 4.1: Zaciemnianie treści z użyciem HTML

Współcześnie do filtrowania spamu coraz częściej używa się sztucznych sieci neuronowych. Stosowane są one chociażby w poczcie Gmail, o czym firma Google poinformowała w lipcu 2015 r., oświadczając przy tym, że skuteczność wykrywania spamu wynosi 99,9% a procent wiadomości treściwych omyłkowo zaklasyfikowanych jako spam wynosi 0,05% [31]. Filtry antyspamowe opierają swe działanie nie tylko na treści e-maila, ale również na metadanych takich jak np. czarne listy adresów DNS.

4.1.1 Analiza wydźwięku

Drugą popularną dziedziną zastosowań klasyfikacji tekstu jest analiza wydźwięku (ang. *sentiment analysis*), czyli analiza emocji zawartych w tekstach. Najczęstszym

celem jest sprawdzenie, czy dany tekst ma wydźwięk pozytywny czy też negatywny. W ogólności techniki analizy sentymentu możemy podzielić na dwie grupy [37] [22] metody słownikowe oraz metody oparte na uczeniu maszynowym.

4.1.1.1 Metody analizy wydźwięku

- **Metody słownikowe**
  Metody te można podzielić na dwie podgrupy:
  - **Oparte na zbiorach słów opiniujących**
    Jednym ze sposobów zgromadzenia kolekcji takich słów jest ręczne stworzenie niewielkiego początkowego zbioru wyrazów, a następnie automatyczne wyszukiwanie ich synonimów i antonimów z wykorzystaniem sieci WordNet. Wadą najprostszej wersji tego podejścia jest nieuwzględnianie kontekstu, w jakim występują słowa opiniujące.
  - **Oparte na korpusach tekstów**
    Używając metod tej klasy wychodzimy od pewnego zbioru słów opiniujących, a następnie staramy się wykryć pewne wzorce związane z występowaniem w korpusie słów opiniujących.
- **Metody oparte na uczeniu maszynowym**
  W przypadku uczenia nadzorowanego, proces przebiegać może w oparciu o techniki, które opisano w podrozdziale 2.4. Istnieje również możliwość wykorzystania technik wspomnianych w podrozdziałach 2.1.2 oraz 2.1.3, które zakładają wykorzystanie danych pozbawionych etykiet (bądź też dokładnych etykiet).

4.1.1.2 Wybór cech w analizie wydźwięku

W procesie analizy wydźwięku ważną rolę odgrywa wspomniana w podrozdziale 2.2 kwestia ustalenia odpowiedniej przestrzeni cech. Do cech ekstrahowanych z tekstu dla potrzeb analizy wydźwięku zaliczamy między innymi [22] [1] :

- **Częstości (lub obecność) termów**
- **Tagi części mowy**
  Uważa się, że szczególne znaczenie dla określenia wydźwięku tekstu mają przymiotniki.

- **Słowa lub frazy opiniujące**
  O wydźwięku tekstu decyduje występowanie termów lub fraz ze słowników wspomnianych w podrozdziale 4.1.1.1.

- **Negacje**
  Obecność negacji (w przypadku języka polskiego głównie: występowanie słowa „nie") zwiększa prawdopodobieństwo negatywnego wydźwięku danej wypowiedzi.

- **Emotikony**
  Emotikony zostały włączone do analizy wydźwięku np. w pracy dotyczącej analizy danych pochodzących z portalu Twitter [1] , w której wykorzystano zbiór 170 emotikonów podzielonych na 5 grup: skrajnie pozytywne, skrajnie negatywne, pozytywne, negatywne oraz neutralne. Przykładowe emotikony przynależne do poszczególnych grup przedstawia tabela 4.1.

Emotikon	**Wynik pozytywny**
:-) :) :o) :] :3 :c)	Pozytywny
:D C:	Skrajnie pozytywny
:-(:(:c :[	Negatywny
D8 D; D= DX v.v	Skrajnie negatywne
:—	Neutralne

Tabela 4.1: Przykłady emotikonów o różnym wydźwięku [1]

- **Wykrzykniki**
  Występowanie wykrzykników podkreśla emocjonalny charakter wypowiedzi.

- **Tekst pisany z wielkiej litery**
  Tekst napisany w całości wielkimi literami można interpretować jako krzyk. Dodatkowo można zastosować *preprocessing* tekstu uwzględniający specyfikę potocznego i emocjonalnego języka. W przytoczonej tu pracy na temat analizy sentymentu wydźwięku wpisów na Twitterze[1] użyto następujących zabiegów:

- **Zamiana akronimów na pełne słowa lub frazy**
  Przykładowo: lol → laughing out loud

- **Transformacja słów zawierających ciągi tych samych liter**
  Wyrazy typu coooool, coool zostały sprowadzone do wspólnej postaci coool, która tworzyła z nich jedną grupę jednocześnie rozróżniając ją od zwykłej formy cool.

4.1.2 Omówienie prac dotyczących filtrowania obraźliwego języka

4.1.2.1 Zastosowanie klasyfikacji wielopoziomowej

W pracy z roku 2010 [28] przedstawiono metodę klasyfikacji wielopoziomowej w celu wykrywania języka obraźliwego. Przeanalizowano korpus wypowiedzi zawierający 1525 przykładów, które zostały podzielone na dwie klasy:

- Dopuszczalne (Okay) – 68%
- Obraźliwe (Flame) – 32%

Dane zostały poddanego *preprocessingowi* polegającemu na usunięciu nagłówków, adresów internetowych i mailowych. Usunięto również między innymi wszystkie cyfry, nawiasy, przecinki i średniki. Pozostawiono natomiast myślniki, kropki, cudzysłowy, znaki zapytania oraz wykrzykniki, które uznano za użyteczne w ustalaniu wydźwięku wypowiedzi. Jako wersję bazową przyjęto klasyfikator zaliczający wszystkie wypowiedzi do najliczniejszej klasy (skuteczność: 68%). Następnie dokonano klasyfikacji za pomocą trzech metod :

1. Naiwny klasyfikator Bayesa, przestrzeń cech: 15 636 wymiarów Dokładność: 84,26% (10 –krotna walidacja krzyżowa), 81,37% (walidacja na odłożonym zbiorze testowym o wielkości 10% korpusu)
2. Naiwny klasyfikator Bayesa, przestrzeń cech: ok. 1 700 wymiarów Dokładność: 90,75% (10 –krotna walidacja krzyżowa), 90,98% (walidacja na odłożonym zbiorze testowym o wielkości 10% korpusu)
3. Naiwny klasyfikator Bayesa, przestrzeń cech: ok. 1 700 wymiarów + klasyfikator regułowy wykorzystujący słownik obraźliwych wyrażeń Dokładność: 96,78% (10 –krotna walidacja krzyżowa), 96,72% (walidacja na odłożonym zbiorze testowym o wielkości 10% korpusu)

Praca ta dowodzi, że redukcja przestrzeni cech może nawet poprawić dokładność klasyfikatora (w eksperymencie klasyfikator wytrenowany na 15 636 wymiarów uzyskał niższą dokładność niż ten wytrenowany na ok. 1700 wymiarów), najlepszy rezultat uzyskano jednak posiłkując się zbiorem reguł danym przez człowieka, a nie – pozyskanym w automatycznym procesie uczenia maszynowego.

4.1.2.2 Zastępowanie wypowiedzi obraźliwych nieobraźliwymi

Pomysł ten został zaprezentowany w pracy dotyczącej cenzurowania komentarzy na portalu YouTube[44] . Korpus wypowiedzi zawierał 11 tysięcy komentarzy z portalu YouTube. W projekcie opisywanym w pracy nie zastosowano uczenia maszynowego. Problem sformułowany w pracy nie dotyczy jednak klasyfikacji całej wypowiedzi, ale zastępowania wyrażeń obraźliwych neutralnymi/nieobraźliwymi z zachowaniem pozostałej części komentarza. Operacja zastępowania słów obraźliwych mogła zostać precyzyjnie przeprowadzona dzięki zastosowaniu analizy składniowej wypowiedzi. Opracowaną metodę wykorzystano w stworzonej wtyczce do przeglądarki Firefox.

4.1.3 Wykrywanie obraźliwych wypowiedzi z zastosowaniem metody "topic modelling"

We wstępie do artykułu autorzy [43], odnosząc się do swoich wcześniejszych prac stwierdzają, że klasyfikacja tweetów na podstawie modelu języka opartego na modelu *bag-of-words* oraz oznaczaniu części mowy nie sprawdziła się ze względu na dużą ilość szumów. Użyto korpusu tweetów zawierającego ponad 680 milionów wypowiedzi. Dane zostały oznaczone metodą *bootstrappingu*, przy czym punktem wyjścia był słownik zawierający 338 słów uznanych za obraźliwe. Sam algorytm wykorzystujący *bootstrapping* działał (w dużym uogólnieniu) w ten sposób, że najpierw za odpowiedzi obraźliwe uznawał tylko te, które zawierały wyrazy ze wspomnianego słownika. W kolejnych iteracjach algorytm wykrywał inne cechy występujące w zgromadzonym na daną chwilę zbiorze wypowiedzi obraźliwych i na ich podstawie wykrywał kolejne, już niezawierające słów ze słownika wyrazów obraźliwych. *Preprocessing* polegał m.in. na usunięciu tweetów nieanglojęzycznych i zredukowaniu zamierzonych powtórzeń trzech lub więcej liter do jednej (przykład: hhhheeeello $\rightarrow$ hello). Przestrzeń cech składała się z dwóch części:

- **Tematy wypowiedzi**
  Cechy te zostały wygenerowane automatycznie w procesie topic modellingu za pomocą algorytmu Latent Dirichlet Allocation (LDA)[4]. Model tematyczny został wytrenowany z użyciem 860 071 tweetów.

- **Cecha leksykalna**
  Cecha binarna, która odnosiła się do tego, czy w wypowiedzi znajduje się co najmniej jeden wyraz z określonego słownika słów obraźliwych. Dzięki zastosowaniu tematów wypowiedzi zamiast podejścia *bag-of-words* udało się uzyskać przestrzeń cech o niewielkiej licznie wymiarów: maksymalna liczba tematów brana pod uwagę w eksperymencie wynosi 50 (co wraz z cechą leskykalną daje łącznie 51 wymiarów).

Zastosowano cztery algorytmy uczenia maszynowego: algorytm tworzenia drzew decyzyjnych J48, regresję logistyczną, maszynę wektorów nośnych oraz las losowy. Jako baseline przyjęto wyniki uzyskane dla klasyfikatora opartego na dopasowywaniu słów kluczowych, który osiągnął wartość *true positive* równą 69,7% i *false positive* równą 3,77%. Autorzy, mimo deklarowanego zastosowania czterech metod uczenia maszynowego, prezentują jedynie rezultaty uzyskane dla regresji logistycznej. Dziesięciokrotna walidacja krzyżowa dla tego algorytmu dała wynik wyższy niż baseline tj. *true posistive* równy 75,1% .

4.1.4 Wykorzystanie paragraph2vec w wykrywaniu wypowiedzi obraźliwych

W pracy zespołu z Yahoo Labs [8] wykorzystano duży zbiór komentarzy pobranych ze strony Yahoo Finance, zawierający 56 280 wypowiedzi obraźliwych oraz 895 456 dopuszczalnych (clean comments). Cały zbiór był pozyskiwany i etykietowany przez 6 miesięcy. Ostatecznie po poddaniu zbioru *preprocessingowi* polegającemu sprowadzeniu wszystkich liter do małej litery, usunięciu znaków specjalnych i słów przestankowych, otrzymano słownik liczący 304 427 termów. Reprezentację wektorową tekstu uzyskano dzięki algorytmowi paragraph2vec . Ten model wektorowej reprezentacji tekstu zestawiono z modelami *bag-of-words TF* oraz *bag-of-words TFIDF*. Porównano trzy rodzaje modeli języka trenując na nich klasyfikator oparty na regresji logistycznej, sprawdzając je za pomocą pięciokrotnej walidacji krzyżowej. Wyniki porównano pod względem

otrzymanego obszaru pod krzywą (ang. area under curve, AUC). W rezultacie dowiedziono, że spośród powyższych trzech najlepsze odwzorowanie tekstu daje model paragraph2vec. Autorzy opublikowanej w 2015 r. pracy stwierdzają, że *„co ciekawe, mimo przemożnego wpływu internetowej mowy nienawiści, według naszej najlepszej wiedzy istnieje niewiele prac odnoszących się do tego problemu”.*

4.1.5 Wykrywanie społeczności iternetowych promujących nienawiść

Praca dotyczące wykrywania społeczności promujących nienawiść na portalu Tumblr [2] ma inny charakter niż pozostałe opisane w tym podrozdziale, gdyż dotyczy wykrywania pewnych struktur kształtujących się na portalach społecznościowych, a dokładniej: wykrywania internetowych społeczności promujących nienawiść. W pracy warto zwrócić uwagę na dwie kwestie:

- Użycie klasyfikacji jednoklasowej (ang. one-class classification, unary classification) Klasyfikator został wytrenowany na zbiorze zawierającym tylko przykłady promowania nienawiści (400 przykładów w zbiorze uczącym)
- Model języka: n-gramy na poziomie znaków

Uzyskano wynik o dokładności 77%, przy czym miara recall wyniosła 86%, a precision – 75%.

ROZDZIAŁ 5

Wybrane narzędzia do pobierania treści z sieci Internet, przetwarzania języka naturalnego oraz uczenia maszynowego

5.1 Narzędzia do pobierania treści z sieci Internet użyte w projekcie magisterskim

W celu pobrania komentarzy ze strony internetowej, w projekcie użyto dwóch modułów dla języka Python:

- **Urllib** Jest to jeden z podstawowych modułów języka Python (udostępniany standardowo z instalacja języka Python). Moduł ten służy do pobierania zasobów sieci Internet.

- **BeautifulSoup** BeautifulSoup jest modułem służącym do parsowania plików HTML. Umożliwia ekstrahowanie określonych treści z pobranej strony internetowej.

5.2 Narzędzia przetwarzania języka naturalnego wykorzystane w projekcie autorskim

5.2.1 Aspell

Aspell jest otwartoźródłowym korektorem pisowni (ang. spell checker). Jego działanie opiera się na dwóch algorytmach [3]:

- **Algorytmie Metaphone Lawrence'a Philipsa**
  Algorytm ten przekształca słowo na ciąg znaków kodujący przybliżoną wymowę słowa wejściowego. Przykład działania odmiany algorytmu Metaphone (zwanej Double-Metaphone) dla języka angielskiego przedstawia tabela 5.1.

Słowo	**Kodowanie w Double Metaphone**
peter	ptr
pete	pt
pedro	ptr

Tabela 5.1: Przykładowe kodowanie w algorytmie Double Metaphone [5]

- **Strategii near miss**
  Dwa słowa są uważane za takie same jeżeli można doprowadzić je do identycznej postaci zamieniając miejscami sąsiednie litery, zmieniając, usuwając, dodając jedną literę lub spację [38].

Zgodnie z bardzo ogólnym opisem umieszczonym na stronie projektu szukanie zbioru słów sugerowanych jako poprawione odbywa się w następujących krokach [3]:

1. Zamiana błędnie zapisanego słowa na reprezentację jego wymowy (Metaphone). Dla uproszczenia nazwijmy reprezentację wymowy słowa wejściowego skrótem RWSW, a słowo wejściowe SW.

2. Wyszukiwanie zbioru słów-kandydatów na podstawie odległości edycyjnej (ang. edit distance) reprezentacji ich wymowy od RWSW. Pod uwagę brane są tylko reprezentacje wymowy odległe od RWSW o nie więcej niż 2. Odległość edycyjna według opisu jest zdefiniowana tak jak odległość Damerau-Levenshteina[41].

3. Dla zbioru słów liczony jest ocena podobieństwa – średnia z odległości edycyjnej słowa kandydata od SW oraz od RWSW.

4. Zwracane jest lista słów, które uzyskały najwyższy wynik w kroku 3.

Aby umożliwić prawidłowe działanie programu Aspell dla języka polskiego, powinniśmy domyślnie używać słownika języka polskiego, który można pobrać na stronie projektu Słownik Języka Polskiego (sjp.pl).[1] Jeżeli chcemy skorzystać z Aspella w skrypcie/programie napisanym w języku Python, wygodnego interfejsu dostarcza nam projekt aspell-python dostępny na portalu GitHub.[2] Przykład działania tego modułu Pythona przedstawia rysunek 5.1. Jak widzimy, dla poprawianego słowa została zwrócona lista sugestii, która powinna być uszeregowana według ich podobieństwa do poprawianego wyrazu. W projekcie przyjęto, że dla korekty błędnie zapisanego słowa automatycznie bierze się pierwszy wyraz z tej listy.

```
#-*- coding: utf-8 -*-
import aspell
speller = aspell.Speller(('lang','pl'),('master', '/home/krzysztof/Pulpit/Projekt/pl/pl.rws'))
suggested = speller.suggest('porpawny')
print suggested
```

```
['poprawny', 'poprawiny', 'poprawnym', 'poprawmy', 'Poprawy', 'oprawny', 'poprany', 'poprawy', 'porwany', 'propan
y', 'porypany', 'poprawna', 'poprawne', 'poprawni', 'poprawn\xb1', 'porywany', 'prawny', 'pospawany', 'poradny', 'p
oranny', 'pora\xbcny', 'pospany', 'porz\xb1dny', 'postawny', 'p\xf3\xb3jawny']
```

Rysunek 5.1: Przykład użycia modułu aspell - poprawianie wyrazu "porpawny"

5.2.2 Morfologik

Projekt Morfologik[3] dostarcza narzędzi (napisanych w języku Java) służących do lemaamtyzacji oraz oznaczania części mowy w języku polskim. Przykładowe znaczniki używane w Morfologiku do opisania rozpoznanych słów przedstawione są tabelach 5.2 i 5.3. W języku Python można posłużyć się interfejsem z projektu pyMorfologik (dostępny w repozytorium na portalu GitHub[4]), który wykorzystuje bibliotekę Morfologik

[1] http://sjp.pl/slownik/ort/
[2] https://github.com/WojciechMula/aspell-python
[3] https://github.com/morfologik
[4] https://github.com/dmirecki/pyMorfologik

Znacznik Morfologika	Część mowy
adj	Przymiotnik
adv	Przysłówek
conj	Spójnik
ign	Ignorowana część mowy
ppron3	Zaimek nietrzecioosobowy
pred	Zaimek trzecioosobowy
subst	Rzeczownik
verb	Czasownik

Tabela 5.2: Morfologik - przykładowe znaczniki części mowy [24]

Znacznik Morfologika	Atrybuty podstawowych form
sg	Liczba pojedyncza
pl	Liczba mnoga
indecl	Forma nieodmienna
dat	Celownik
pos	Stopień równy
n	Rodzaj nijaki
aff	Forma niezanegowana
impt	Tryb rozkazujący
bedzie	Forma przyszła "być"
praet	Forma przeszła czasownika (pseudoimiesłów)

Tabela 5.3: Morfologik - przykładowe atrybuty form podstawowych [24]

w wersji 1.9 . Przykładowe użycie modułu pyMorfologik pokazano na rysunku 5.2. Jak widać, obiekt klasy przyjmuje na wejściu listę list, przy czym każda zawiera pojedynczy dokument. Nie jest potrzebna dodatkowa lematyzacja na wejściu – pyMorfologik rozbija zdanie na pojedyncze wyrazy po spacjach. Moduł ten jest czuły na wielkość znaków oraz na wszelkie błędy w pisowni, nie radzi sobie także z z wielkimi literami na początku zdania.

```
from pyMorfologik import Morfologik
from pyMorfologik.parsing import ListParser

parser = ListParser()
stemmer = Morfologik()
output = stemmer.stem(['Ala ma kota, a kot ma Alę.'],parser)
print output
```

```
[(u'Ala', {u'Alo': [u'subst:sg:acc:m1+subst:sg:gen:m1'], u'Al': [u'subst:sg:acc:m1+subst:s
g:gen:m1'], u'Ala': [u'subst:sg:nom:f']}), (u'ma', {u'm\xf3j': [u'adj:sg:nom.voc:f:pos'],
u'mie\u0107': [u'verb:fin:sg:ter:imperf:refl.nonrefl']}), (u'kota', {u'kot': [u'subst:sg:ac
c:m1'], u'kota': [u'subst:sg:nom:f']}), (u'a', {u'a': [u'conj+interj+prep:nom+qub']}), (u'k
ot', {u'kot': [u'subst:sg:nom:m1'], u'kota': [u'subst:pl:gen:f']}), (u'ma', {u'm\xf3j':
[u'adj:sg:nom.voc:f:pos'], u'mie\u0107': [u'verb:fin:sg:ter:imperf:refl.nonrefl']}), (u'Al
\u0119.', {})]
```

Rysunek 5.2: Przykład użycia modułu pyMorfologik do lematyzacji i oznaczenia części mowy w zdaniu "Ala ma kota, a kot ma Alę"

5.3 Narzędzia uczenia maszynowego użyte w projekcie

5.3.1 Scikit-learn

Scikit-leran jest pakietem do uczenia maszynowego napisanym w jezykujęzyku Python. Zawiera narzędzia służące do klasyfikacji, regresji, klastrowania, redukcji wymiarów a także do preprocessingu danych (np. do przetwarzania tekstu takich jak tokenizator dla języka angielskiego). Spośród dostępnych klasyfikatorów, w projekcie opisawanymopisywanym w tej pracy używane są:

- **Naiwny klasyfikator Bayesa**
  Moduł sklearn.naive bayes: GaussianNB
- **N-najbliższych sąsiadów**
  Moduł sklearn.neighbors: KneighborsClassifier, NearestCentroid
- **Maszyna wektorów nośnych**
  Moduł sklearn.svm : SVC, LinearSVC.

Ponadto w pakiecie scikit-learn (sklearn) znajdują się implementacje wielu innych algorytmów uczenia maszynowego (nazdzorowanego, nienadzorowanego i półnadzorowane-

go) jak np. drzewa decyzyjne czy maszyna Boltzmanna. W pakiecie znajdujemy również inne przydatne narzędzia, spośród których w projekcie użyto dwóch:

- **KFold** – narzędzie służące do podziału zbioru dla celów walidacji krzyżowej. Generuje dwie listy indeksów dzielące korpus na zbiór uczący oraz zbiór testowy.
- **TfidfTransformer** – narzędzie służące do przekształcania zbioru wektorów reprezentujących częstość wystąpień termów w dokumentach na wektory ważone metodą tfidf.

Użycie modułu pakietu Scikit-learn na przykładzie trenowania klasyfikatora SVC (maszyna wektorów nośnych) testowanego walidacją krzyżową przedstawia rysunek 5.3.

```
Y = np.array(y).ravel()
X = np.array(x).reshape(len(x),len(x[0]))
kf = KFold(len(x), n_folds=5)

#Trenowanie
from sklearn import svm
svmModel = svm.SVC()

#Testowanie
results=[]
#Trenowanie & testowanie
for train, test in kf:
    svmModel.fit(X[train], Y[train])
    results.append(accuracy_score(Y[test], svmModel.predict(X[test])))

print "Walidacja krzyżowa (5-krotna):", np.mean(results)
```

Rysunek 5.3: Przykład użycia narzędzi z modułu Scikit-learn

5.3.2 Theano/Keras

Theano to framework dla języka Python służący do budowania do sztucznych sieci neuronowych. Wysoką wydajność Theano zapewnia dzięki dynamicznemu generowaniu modeli w języku C[11]. Theano pozwala również na prowadzenie obliczeń na karcie graficznej. Keras jest frameworkiem dla języka Python, pozwalającym na łatwe tworzenie sieci neuronowych z jednym z dwóch frameworków: Theano lub Tensorflow . Keras umożliwia szybkie budowanie różnego rodzaju sieci neuronowych, np. jednokierunkowych, rekurencyjnych, konwolucyjnych. Przykład zastosowania pakietów Keras i Theano w celu skonstruowania sieci jednokierunkowej pokazany jest na rysunku 5.4.

```
kf = KFold(len(nX), n_folds=5)

#Stworzenie sieci neuronowej
model = Sequential()
model.add(Dense(1024, input_shape=(inp2,)))
model.add(Activation('relu'))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(2))
model.add(Activation('softmax'))

rms = RMSprop()
model.compile(loss='categorical_crossentropy', optimizer=rms)

wagi = model.get_weights()
results = []

#Walidacja krzyżowa
for train, test in kf:
    model.set_weights(wagi)
    model.fit(X[train], Y[train], nb_epoch=2, batch_size=32)
    score = model.evaluate(X[test], Y[test], show_accuracy=True, verbose=0)
    print 'score:', score[1]
    resultsExt.append(score[1])

print "Walidacja krzyżowa", np.mean(results)
```

Rysunek 5.4: Sieć jednokierunkowa (Theano+Keras)

5.3.3 Gensim

Gensim jest pakietem do przetwarzania języka naturalnego dla języka Python. Zawiera on między innymi narzędzia wykorzystywane w metodzie *topic modelling*. Pod pojęciem *topic modelling* rozumiemy automatyczne określanie tematyki dokumentów należących do pewnego korpusu. W pakiecie możemy znaleźć także implementacje algorytmów word2vec oraz doc2vec opisanych w częściach 2.2.2.4 i 2.2.2.5. Przykłady użycie ich użycia zostały zaprezentowane na rysunkach 5.5 i 5.6. Implementacja algorytmu word2vec przyjmuje na wejściu listę list tokenów znajdujących się w dokumentach należących do danego korpusu. W doc2vec musimy z kolei dodatkowo dostarczyć na wejściu listę etykiet przyporządkowanych wszystkim dokumentom.

```
from gensim.models import word2vec
lemmas = [doc.lemmas for doc in model.documents]
num_features = 500
min_word_count = 1
num_workers = 4
context = 15
w2v = word2vec.Word2Vec(lemmas, workers=num_workers,size=num_features, min_count = min_word_count, window = context)
w2v.most_similar(u'prezydent')
```

```
[(u'duda', 0.999984622001648),
 (u'Kaczy\u0144ski', 0.9999808669090271),
 (u'brawo', 0.9999808669090271),
 (u'wa\u0142\u0119sa\u0107', 0.99997878074646),
 (u'Tusk', 0.9999784231185913),
 (u'powinien', 0.9999784231185913),
 (u'nasz', 0.9999780654907227),
 (u'prezes', 0.9999775886535645),
 (u'd\u0119bski', 0.9999772310256958),
 (u'premiera', 0.9999771118164062)]
```

Rysunek 5.5: Przykład użycia narzędzia word2vec z pakietu Gensim

```
from gensim.models import doc2vec
#Generowanie etykiet dokumentów
labels = ['SENT_'+str(ind) for ind in range(len(lemmas))]
ls = [doc2vec.LabeledSentence(lemm,[label]) for lemm, label in zip(lemmas,labels)]
min_count=1
window=10
size=500
sample=1e-4
negative=5
workers=4
d2v = doc2vec.Doc2Vec(size=size, window=window, min_count=min_count, alpha=0.025, min_alpha=0.025, workers=4)
d2v.build_vocab(ls)
d2v.most_similar(u'Polska')
```

```
[(u'w\u0105ski', 0.18413211405277252),
 (u'jednoznacznie', 0.16454505920410156),
 (u'wyliczy\u0107', 0.15617762506008148),
 (u'mi\u0142o\u015bnik', 0.14979633688926697),
 (u'nagrobek', 0.1497402787208557),
 (u'czerp', 0.1463025063276291),
 (u'Lessing', 0.14252494275569916),
 (u'cudowny', 0.13960148394107819),
 (u'Kiero\u0144ski', 0.13910658657550812),
 (u'cacanka', 0.13864865899085999)]
```

Rysunek 5.6: Przykład użycia narzędzia doc2vec z pakietu Gensim

ROZDZIAŁ 6

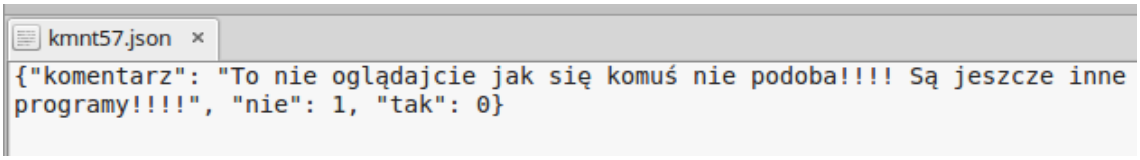
Projekt magisterski

6.1 Gromadzenie korpusu

W projekcie użyto zgromadzonego od podstaw zbioru komentarzy (dalej zwanego korpusem *Komentarze*). Został on pozyskany z portalu Wirtualna Polska, a w szczególności z następujących jego działów:

- Wiadomości (wiadomosci.wp.pl)
- Sport (sport.wp.pl)
- Teleshow (teleshow.wp.pl)

Komentarze pobrano automatycznie za pomocą autorskiego programu typu *crawler* napisanego w języku Python z wykorzystaniem modułów urllib (pobieranie zasobów z sieci www) oraz BeautifulSoup (parsowanie plików html). Każdy komentarz został zapisany w osobnym pliku w formacie JSON.



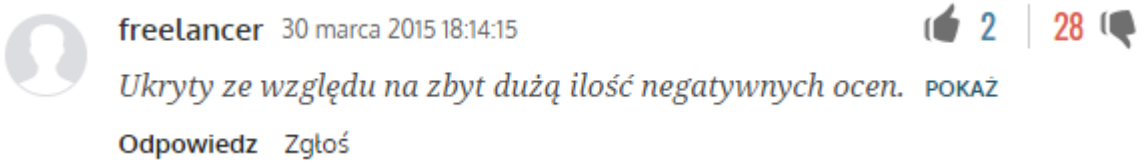
Rysunek 6.1: Przykładowy komentarz z korpusu *Komentarze*

Jak widzimy na rysunku 6.1, każdy z plików JSON zawierał trzy składowe:

- Treść komentarza (wartość dla klucza „komentarz”)

- Liczba ocen „na nie” (wartość dla klucza „nie”)
- Liczba ocen „na tak” (wartość dla klucza „tak”)

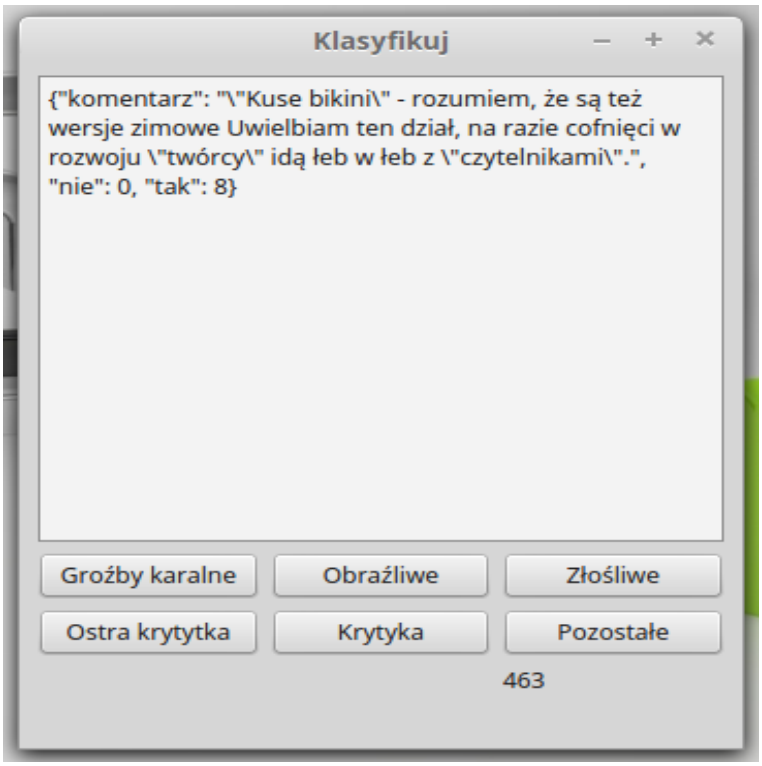
Dzięki wybraniu jednego źródła komentarzy, udało się zachować spójną strukturę plików JSON (różne portale w różny sposób prezentują wyniki oceny komentarza przez internautów – np. jako procent ocen „na nie” i „na tak”). Początkowo brana była pod uwagę możliwość wykorzystania istniejących już etykiet komentarzy w procesie uczenia maszynowego . Dzięki opcji oceniania komentarzy, internauci mogą każdemu z nich przyznać oceną „na tak” lub „na nie” - informacja ta jest później wykorzystywana np. w systemie pozycjonowania komentarzy, tzn. można przeglądać komentarze posortowane według kryterium „najwyżej ocenione”. Przykład działania takiego systemu widzimy na rysunku 6.2.



Rysunek 6.2: Przykładowy komentarz ze strony funduszeue.wp.pl [1]

Pierwsza analiza zbioru pozyskanych komentarzy kazała jednak stwierdzić, iż nie można w łatwy sposób posortować komentarzy na obraźliwe i nieobraźliwe tylko na podstawie oddanych głosów „na tak” i „na nie”. W związku z tym dokonano ręcznej klasyfikacji komentarzy za pomocą autorskiego programu napisanego w języku Python (rysunek 6.3). Program posiada graficzny interfejs użytkownika, a jego działanie polega na wyświetlaniu niesklasyfikowanych plików JSON i przenoszeniu ich do odpowiednich folderów w zależności od wyboru dokonanego przez użytkownika. Przykładowo, po kliknięciu przycisku „Złośliwe” wyświetlany aktualnie komentarz zostaje przeniesiony do folderu „Złośliwe”, a w oknie programu automatycznie pojawia się nowy, niesklasyfikowany komentarz.

[1] `http://funduszeue.wp.pl/informacje/programisci-pomoga-gospodarce-rusza-projekt-epionier_s99?utm_source=WP_wiadomosci&utm_medium=WP_wiadomosci_modul3&utm_content=promo&utm_campaign=wp_support`



Rysunek 6.3: Ręczne sortowanie komentarzy

Chociaż wynikiem końcowym miał być klasyfikator binarny (Obraźliwe-Nieobraźliwe) lub trójklasowy (Obraźliwe – Klasa pośrednia – Nieobraźliwe), roboczo przyjęto podział na 6 kategorii:

- **Groźby karalne**
  Komentarze zawierające groźby karalne, życzenie śmierci i inne komentarze uznane za szczególnie obraźliwe.

- **Obraźliwe**
  Komentarze bezpośrednio obrażające pewną osobę lub grupę osób, w tym także szczególnie niekulturalne uwagi na temat czyjegoś wyglądu, o podtekście seksualnym itp.

- **Złośliwe**
  Komentarze obrażające pewną osobę lub grupę osób w sposób pośredni.

- **Ostra krytyka**
  Komentarze zawierające ostre sformułowania

- **Krytyka**
  Nieobraźliwe komentarze zawierające negatywne opinie.

- **Pozostałe**
  Nieobraźliwe komentarze zawierające neutralne bądź pozytywne opinie.

Podział korpusu na większą liczbę klas został dokonany celowo, aby umożliwić przesuwanie granicy między poszczególnymi klasami. Wartości klas w korpusie *Komentarze* stanowią pewne continuum, którego dwoma skrajnymi obszarami są klasy „Groźby karalne" oraz „Pozostałe". O ile więc niemożliwe jest, aby komentarz o treści kwalifikującej go do grupy „Groźby karalne" znalazł się w grupie „Pozostałe", o tyle możliwe jest przesunięcie komentarza do klasy sąsiedniej. W oznaczaniu komentarzy wzięły udział dwie osoby, przy czym prawie 97% korpusu zostało oznaczone przez autora tego projektu. Liczbowe podsumowanie korpusu *Komentarze* przedstawiono w tabeli 6.1.

Kategoria	**Liczba**
Groźby karalne	43
Obraźliwe	1970
Złośliwe	1474
Ostra krytyka	808
Krytyka	598
Pozostałe	2465
Łącznie:	**7358**

Tabela 6.1: Statystyki korpusu *Komentarze*

Cały korpus jest dostępny publicznie w repozytorium na portalu Github[2] . Dysponując posegregowanym korpusem, sprawdzono tezę, czy istnieje jakakolwiek zależność między ocenami internautów a etykietami nadanymi komentarzom na potrzeby niniejszej pracy. Zredukujmy 6-klasowy podział do nowego, 3-klasowego według klucza:

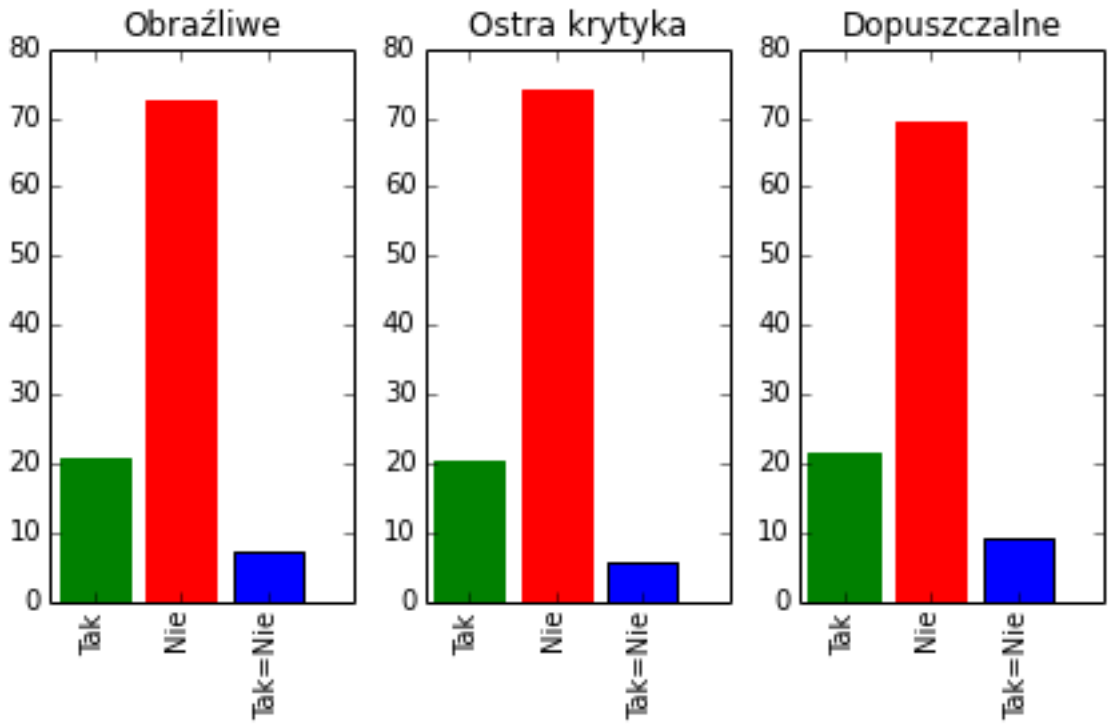
- Obraźliwe (Groźby karalne, Obraźliwe, Złośliwe)
- Ostra krytyka (Ostra krytyka)
- Dopuszczalne (Krytyka, Pozostałe)

[2] `https://github.com/krzjoa/Komentarze`

Najłatwiejszym sposobem na automatyczne posegregowanie komentarzy na trzy klasy wydaje się proste wykorzystanie głosowania poprzez podział według kryteriów:

- Tak – komentarze ocenione przez większość internautów na tak
- Nie – komentarze ocenione przez większość internautów na nie
- Tak=Nie – komentarze, którym przyznano równą liczbę komentarzy na tak i na nie

Rozkład ocen w poszczególnych klasach obrazują wykresy na rysunku 6.4.

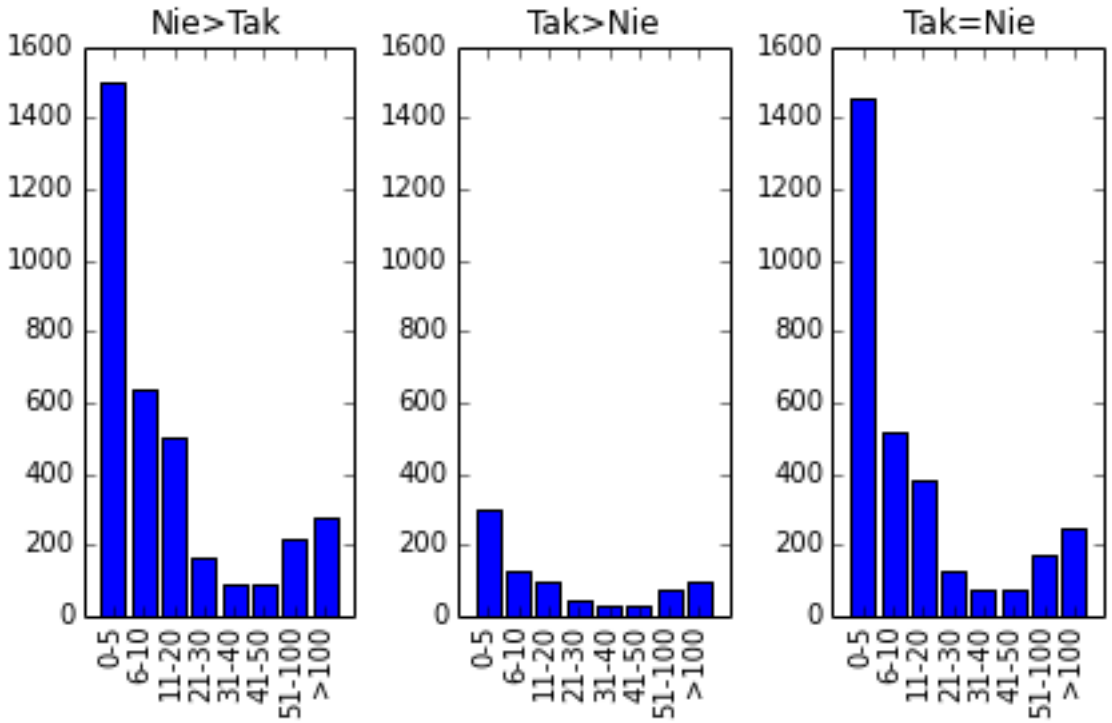


Rysunek 6.4: Procentowy rozkład ocen komentarzy w klasach korpusu *Komentarze*

Przyjmijmy automatyczny podział według ocen komentarzy:

- Obraźliwe – komentarze ocenione przez większość internautów na nie
- Ostra krytyka – komentarze, które otrzymały tę samą liczbę głosów „na tak” oraz „na nie”
- Dopuszczalne – komentarze, które otrzymały większość głosów „na tak”

Jak widzimy na wykresach na rysunku 6.4, sortując komentarze według ich ocen nie otrzymalibyśmy korpusu zbliżonego do korpusu *Komentarze*. Przykładowo automatycznie wybrana klasa Obraźliwe zawierałaby ok. 70% komentarzy każdej klasy z korpusu *Komentarze*. Zbadano również hipotezę, czy któraś z kategorii w korpusie *Komentarze* nie jest częściej oceniana. Można bowiem przypuszczać, że istnieje tego typu prawidłowość np. komentarze obraźliwe będą częściej oceniane jako kontrowersyjne, wzbudzające więcej emocji. Liczba komentarzy o liczbie ocen należącej do jednego z 8 przedziałów (0-5, 6-10, 11-20, 21-30, 31-40, 41-50, 51-100, powyżej 100) przedstawiona jest na rysunku 6.5.



Rysunek 6.5: Liczba ocen komentarzy

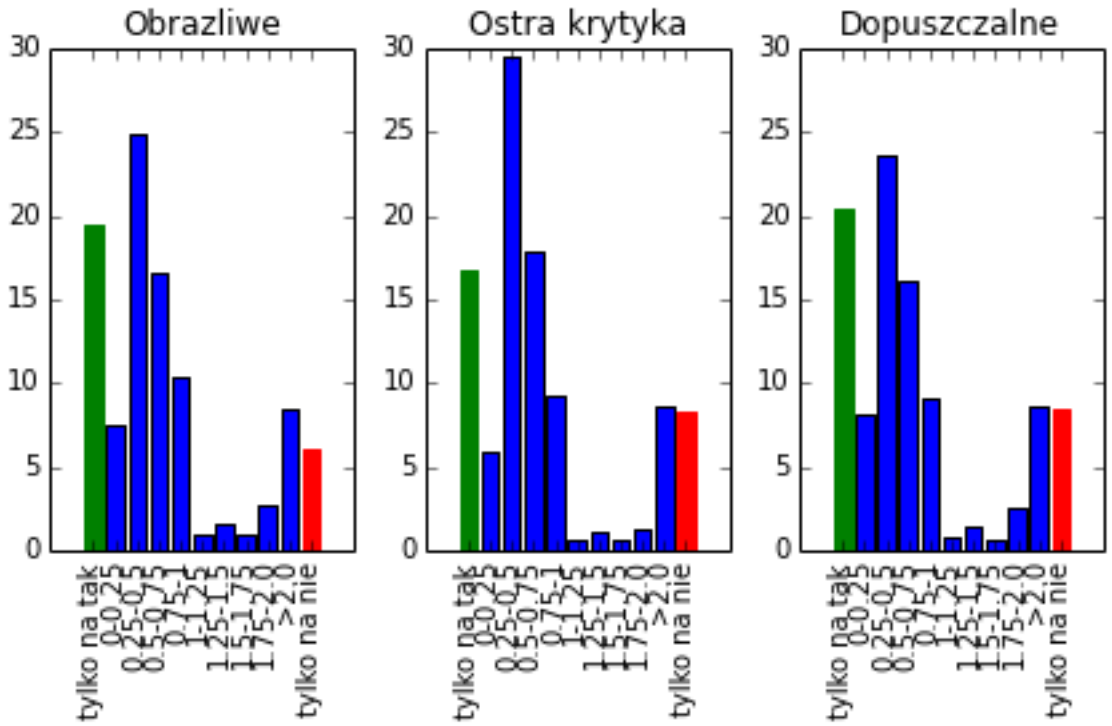
Średnia liczba ocen na komentarz wynosi odpowiednio:

- **Obraźliwe**: 46,8610447252
- **Ostra krytyka**: 58,5555555556
- **Dopuszczalne**: 55,0386996904

Ostatnim testem, jakiemu poddano nieprzetworzony korpus *Komentarze* było sprawdzenie, czy poszczególne klasy nie charakteryzują się specyficznym stosunkiem ocen negatywnych do pozytywnych. Przyjęto 10 możliwych klas:

- Oceny tylko na tak
- Stosunek nie/tak $\langle 0; 0.25\rangle$
- Stosunek nie/tak $(0.25; 0.5\rangle$
- Stosunek nie/tak $(0.5; 0.75\rangle$
- Stosunek nie/tak $(0.75; 1\rangle$
- Stosunek nie/tak $(1; 1.25\rangle$
- Stosunek nie/tak $(1.25; 1.5\rangle$
- Stosunek nie/tak $(1.5; 1.75\rangle$
- Stosunek nie/tak $(1.75; 2\rangle$
- Stosunek nie/tak > 2
- Oceny tylko na nie

Wyniki dla korpusu *Komentarze* przedstawia rysunek 6.6.
Uzyskane wyniki dowodzą, że manualna ocena komentarzy było konieczna, ponieważ rozkład ocen internautów ma bardzo niewielki związek podziałem na klasy przyjętym w prezentowanym tutaj projekcie.



Rysunek 6.6: Stosunek ocen na tak i na nie (procent w całym korpusie)

6.2 Ekstrakcja cech pojedynczego komentarza

W celu przetworzenia zbioru komentarzy na zbiór wektorów, w języku Python napisany został moduł text2vec wykorzystujący pakiet Morfologik (za pośrednictwem interfesju z pyMorfologik) do lematyzacji i oznaczania części mowy oraz spell-checker Aspell (poprzez aspell-python) do automatycznego poprawiania tekstów. Podczas przetwarzania tekstu za pomocą modułu text2vec, każdy komentarz przechowywany był w obiekcie zwierającym:

- **Oryginalną treść komentarza**
- **Wektor znaczników cech szczególnych**
  - **Liczba ciągów znaków napisanych w całości wielkimi literami**
  - **Liczba wielokropków**
    Poprzez wielokropki rozumiane są ciągi kropek o długości co najmniej 2

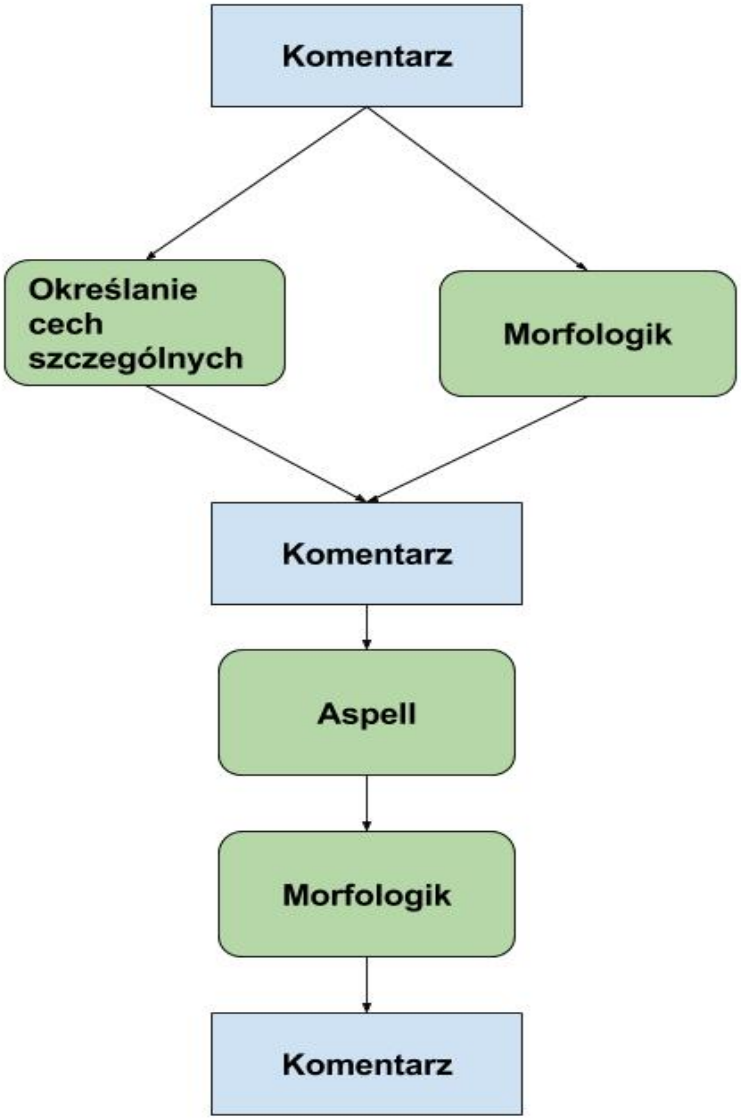
- Liczba ciągów składających się wykrzykników i znaków zapytania Ciągi znaków długości minimum 2 składające się z wykrzykników i znaków zapytania, np. : „!!”, „??” albo „?!?!?!??!??????”.
- **Liczba ciągów znaków zapisanych w cudzysłowie**

- **Listę lematów**
- **Listę bigramów lematów**
- **Listę tagów części mowy**
- **Listę bigramów tagów części mowy**
- **Listę słów nierozpoznanych**

Schemat przetwarzania komentarzy przedstawia rysunek 6.7.
Etapy przetwarzania tekstu:

1. Wstępne przetwarzanie
   (a) Określanie znaczników cech szczególnych (wyrażenia regularne) *Preprocessing* tekstu
      i. Sprowadzenie wszystkich liter do małej (lowercasing)
      ii. Zastąpienie przecinków i kropek spacjami
      iii. Usunięcie wszystkich pozostałych znaków interpunkcyjnych
   (b) Przetwarzanie tekstu za pomocą pakietu Morfologik

      W pierwszym etapie słowa nie są automatycznie poprawiane przez Aspell, aby uniknąć „poprawiania” słów, które są już prawidłowo zapisane. Morfologik do poprawnego działania musi otrzymać słowo zapisane prawidłowo – nawet błędne użycie małej litery (np. „poznań”) spowoduje, że Morfologik nie zwróci żadnej odpowiedzi. Na tym etapie wszelkie wyrazy o błędnej pisowni trafiają na listę słów nierozpoznanych.
2. Poprawianie wyników przetwarzania
   (a) Poprawianie słów nierozpoznanych na etapie 1 (Aspell)
   (b) Przetwarzanie poprawionych słów za pomocą pakietu Morfologik

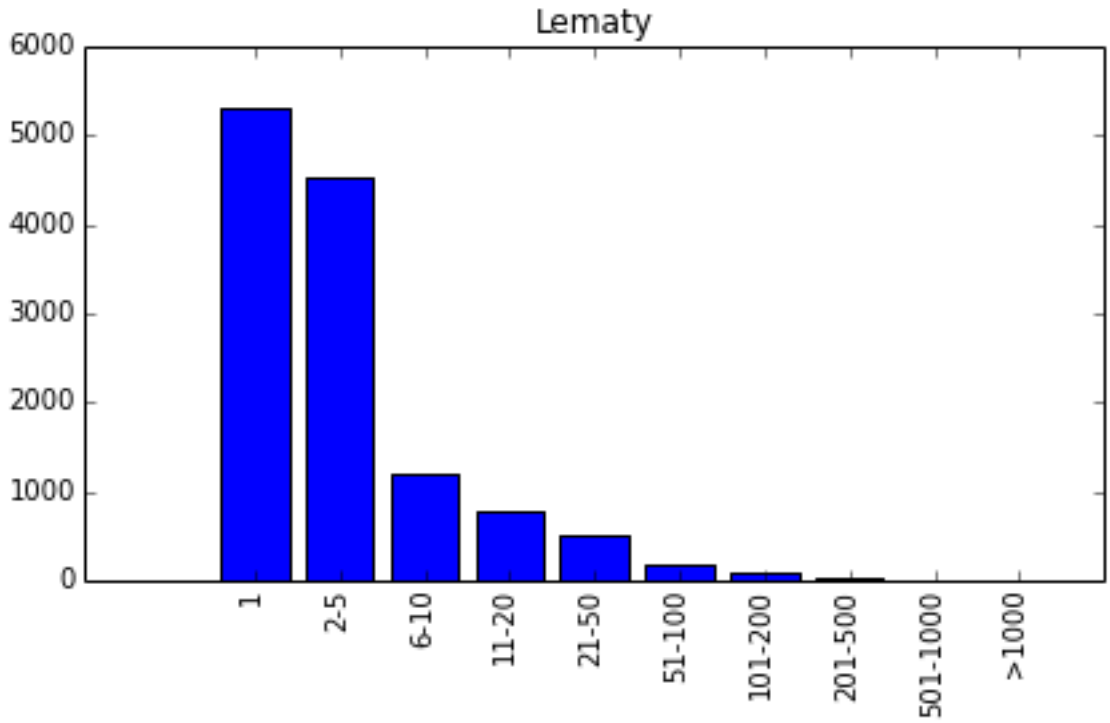


Rysunek 6.7: Schemat przetwarzania komentarza

W wyniku przetworzenia 7325 komentarzy uzyskano bazowy model języka o charakterystyce przedstawionej w tabeli 6.2. Jak widać, wykorzystanie wszystkich wyodrębnionych unikatowych lematów, bigramów lematów, POS-tagów oraz bigramów POS-tagów wymagałoby zastosowania przestrzeni o ponad 100 000 wymiarach,co znacznie utrudniłoby obliczenia. Dodatkowo przeanalizowano liczebność wystąpień poszczególnych termów, co zostało przedstawione na wykresach na rysunkach 6.8 i 6.9.

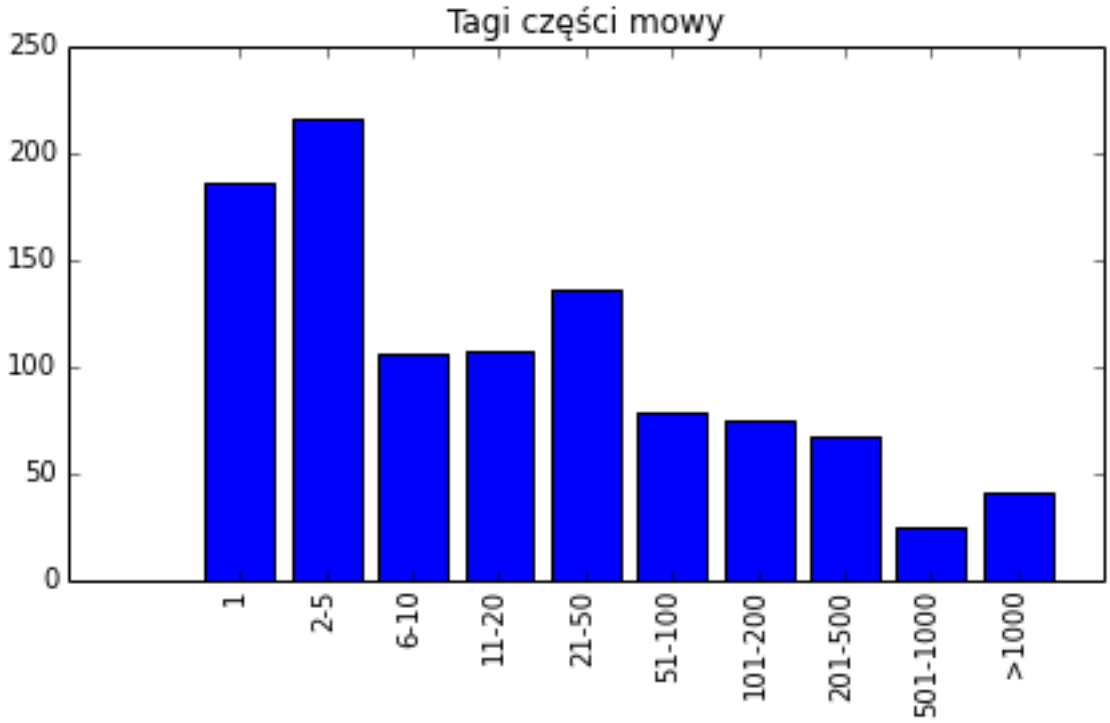
Charakterystyka	Liczba
Unikalne lematy	12075
Unikalne bigramy lematów	80636
Unikalne POS-tagi	1043
Unikalne bigarmy POS-tagów	32181
Unigramy ogółem	151776
Bigramy ogółem	144418
Nierozpoznane unigramy ogółem	354
Unikalne nierozpoznane unigramy	294

Tabela 6.2: Charakterystyki przetworzonego korpusu *Komentarze*



Rysunek 6.8: Liczebność lematów o danej liczbie wystąpień

Ostatecznie, w wyniku przetwarzania, każdemu komentarzowi zostaje przypisany wektor w postaci przedstawionej na rysunku 6.10.



Rysunek 6.9: Liczebność tagów części mowy o danej liczbie wystąpień

[*znaczniki*|*lematy*|*tagi części mowy*]

Rysunek 6.10: Wektorowa reprezentacja komentarza

Wektor ten składa się z trzech części:

- **Znaczniki cech szczególnych**
- **Lematy**
  Podwektor, którego długość zależy od liczby unikalnych lematów w zbiorze uczącym.
- **Tagi części mowy**
  Podwektor, którego długość zależy o liczby unikalnych znaczników części mowy w zbiorze uczącym .

6.3 Porównanie metod uczenia maszynowego

6.3.1 Wybór testowanych metod

Celem tej części projektu było przede wszystkim znalezienie optymalnej przestrzeni cech, tj. spełniającej w miarę możliwości jednocześnie dwa warunki:

1. Maksymalizacja dokładności klasyfikatorów
2. Minimalizacja liczby wymiarów przestrzeni cech

W pierwszej fazie projektu przyjęto model wektorowy przedstawiony w podrozdziale 6.2 i sprawdzono wyniki dla czterech typów metod uczenia maszynowego.

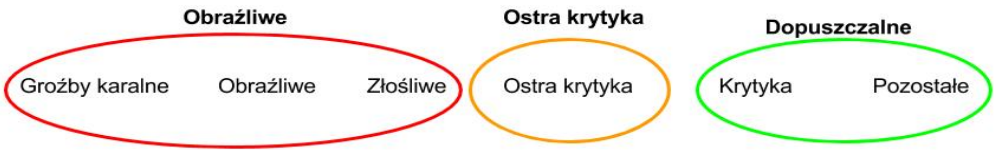
- **Naiwny klasyfikator Bayesa**
  - Gaussian Naive Bayes (scikit-learn)
  - Komplementarny naiwny klasyfikator Bayesa (implementacja własna)[3]
  - Negacyjny naiwny klasyfikator Bayesa (implementacja własna)[3]
- **K-najbliższych sąsiadów**
  - KNeighbors Classifier (scikit-learn)
  - Nearest Centroid (scikit-learn)
- **Maszyna wektorów nośnych**
  - SVC (scikit-learn)
  - Linear SVC (scikit-learn)
- **Sieć neuronowa**
  - Sequential model (Keras+Theano)

[3] `https://github.com/krzjoa/Bayes`

6.3.2 Podział na klasy

Klasyfikatory zostały przetestowane dla dwóch problemów klasyfikacji: binarnej i trójklasowej.

- **Klasyfikacja trójklasowa**
  W klasyfikacji trójklasowej przyjęto podział korpusu według schematu przedstawionego na rysunku 6.11. Wykorzystano wszystkie dokumenty z korpusu *Komentarze*. Rozkład liczebności poszczególnych klas przedstawiono w tabeli 6.3. Jako wersję baseline przyjęto naiwny klasyfikator zaliczający wszystkie nowe przykłady do najliczniejszej kategorii, a więc charakteryzujący się dokładnością równą 47,39%.



Rysunek 6.11: Podział na klasy dla klasyfikacji trójklasowej

Klasa	Liczebność klasy	Procent w korpusie
Obraźliwe	3487	47,39%
Ostra krytyka	808	10,98%
Dopuszczalne	3063	41,63%
Łącznie	7358	100%

Tabela 6.3: Rozkład liczebności klas dla klasyfikacji trójklasowej

- **Klasyfikacja binarna**

  W klasyfikacji binarnej przyjęto nowy podział na klasy, który tym razem nie wykorzystywał wszystkich klas zawartych w korpusie a jedynie jego skrajne klasy. Rozkład liczebności dla klasyfikacji binarnej przedstawiono w tabeli 6.4. Za wersję baseline przyjęto klasyfikator o dokładności 55,05%.

Rysunek 6.12: Podział na klasy dla klasyfikacji binarnej

Klasa	Liczebność klasy	Procent w korpusie
Obraźliwe	2013	44,95%
Dopuszczalne	2465	55,05%
Łącznie	**4478**	**100%**

Tabela 6.4: Rozkład liczebności klas dla klasyfikacji binarnej

6.3.3 Klasyfikacja trójklasowa

Na samym początku sprawdzono 8 różnych algorytmów, testując wszystkie w problemie klasyfikacji trójklasowej na wektorach komentarzy składających się z 3 podwektorów: znaczników cech szczególnych, lematów oraz wskaźników części mowy. Wyniki eksperymentu przedstawiono w tabeli 6.5.

Dla dalszych testów wyselekcjonowano metody, które najlepiej poradziły sobie na pierwszym etapie. Po pojęciami „naiwny klasyfikator Bayesa, k-najbliższych sąsiadów, maszyna wektorów nośnych, sieć neuronowa" użytymi w kolejnych podrozdziałach rozumie się więc metody zaprezentowane w tabeli 6.6. W następnej kolejności sprawdzono, które z części wektora modelu bazowego niosą najwięcej informacji. Tym razem podczas treningu użyto tylko tych wersji algorytmów, które okazały się najskuteczniejsze dla wersji baseline. Wyniki treningu klasyfikatorów na wektorach niezawierających podwektora znaczników cech szczególnych przedstawia tabela 6.7. Wyniki porównano

Metoda	Wersja algorytmu	Parametry	Dokładność
Naiwny klasyfikator Bayesa	Gaussian Naive Bayes	Brak	0,480699501966
	Komplementarny naiwny klasyfikator Bayesa	Brak	0,37578159637
	Negacyjny naiwny klasyfikator Bayesa	Brak	0,352813636095
K-najbliższych sąsiadów	Nearest Centroid	Domyślne	0,427419605119
		Shrink_treshold=5	0,370204016788
		Shrink_treshold=10	0,4739038993
	KNeighbors Classifier	k=5	0,520519352407
Maszyna wektorów nośnych	SVC	SVC	0,474857480123
	Linear SVC	Domyślne	0,653846651199
Sieć neuronowa	Jednokierunkowa sieć neuronowa (Keras Sequential model)	Warstwa ukryta złożona z 2 warstw po :1024 i 512 jednostek Algorytm optymalizacji: RMSProp Rozmiar wsadu: 32 Liczba epok: 2	0,650181405728

Tabela 6.5: Porównanie dokładności klasyfikacji dla różnych metod uczenia maszynowego

Metoda	Wersja algorytmu
Naiwny klasyfikator Bayesa	Gaussian Naive Bayes z pakietu Scikit-learn
K-najbliższych sąsiadów	KNeighbors Classifier z pakietu Scikit-learn (k=5)
Maszyna wektorów nośnych	LinearSVC z pakietu Scikit-learn
Sieć neuronowa	Jednokierunkowa sieć neuronowa (Keras + Theano) Warstwa ukryta złożona z 2 warstw po :1024 i 512 jednostek Algorytm optymalizacji: RMSProp Rozmiar wsadu: 32 Liczba epok: 2

Tabela 6.6: Wyselekcjonowane metody uczenia maszynowego

Metoda	Dokładność
Naiwny klasyfikator Bayesa	0,481243996246
K-najbliższych sąsiadów	0,473906208436
Maszyna wektorów nośnych	0,64129216392
Sieć neuronowa	0,65439336225

Tabela 6.7: Dokładność klasyfikacji - lematy + tagi części mowy

z dokładnością klasyfikatorów wytrenowanych jedynie na wektorach złożonych z podwektora lematów (tabela 6.8).

Metoda	Dokładność
Naiwny klasyfikator Bayesa	0,53112355171
K-najbliższych sąsiadów	0,4739038993
Maszyna wektorów nośnych	0,647592956581
Sieć neuronowa	0,652491650164

Tabela 6.8: Dokładność klasyfikacji - lematy

Jak wskazują wyniki, w przypadku prezentowanego w tej pracy problemu klasyfikacji wypowiedzi w języku polskim największe znaczenie lematy i nic nie zyskujemy dodając informację na temat dodatkowych cech tekstu czy tagów części mowy.Jedynie sieć neuronowa uzyskała nieznacznie wyższą dokładność przy uwzględnieniu innych cech. Ze względu na zbyt duży rozmiar wektorów, zrezygnowano z testów testowania wyników dla wektorów bigramów lematów.

6.3.4 Klasyfikacja binarna - klasy skrajne

Kolejna próba polegała na wytrenowaniu klasyfikatorów binarnych z użyciem najlepszej reprezentacji wektorowej spośród opisanych w poprzednim rozdziale tj. opartej wyłącznie na wektorach lematów. Jak widać w tabeli 6.9, mimo, że tym razem klasyfikatory były trenowane na mniejszej liczbie przykładów, zmniejszenie ilości klas oraz

oparcie się wyłącznie na zbiorach złożonych z klas skrajnych (pominięto klasy środkowe: Złośliwe, Ostra krytyka i Krytyka) spowodowało wzrost ich dokładności .

Metoda	Dokładność
Naiwny klasyfikator Bayesa	0,649408669194
K-najbliższych sąsiadów	0,6152309457
Maszyna wektorów nośnych	0,784502693536
Sieć neuronowa	0,7811524840384001

Tabela 6.9: Klasyfikacja dwuklasowa - klasy skrajne

6.3.5 Klasyfikacja z użyciem wektorów termów ważonych metodą TFIDF

Jak wykazano w podrozdziale 6.3.3, użycie tagów części mowy oraz znaczników cech tekstu dla potrzeb klasyfikacji wypowiedzi z korpusu *Komentarze* nie poprawiło dokładności klasyfikatorów, a nawet spowodowało ich nieznaczne pogorszenie w stosunku do klasyfikatorów wytrenowanych na wektorach określających jedynie występowanie lematów w tekście. W podrozdziale 6.3.4 sprawdzono, jak na dokładność klasyfikatorów wpłynie zmiana wejścia na wektory ważone metoda TFIDF. Transformacja wektorów lematów na wektory lematów ważone TFIDF została przeprowadzona za pomocą dostępnego pakiecie Scikiti-learn narzędzia TfidfTransformer. Wyniki klasyfikatorów uzyskane po wytrenowaniu ich na nowym zestawie wektorów przedstawiono w tabeli 6.10. Jak widać, użycie ważenia TFIDF wpłynęło w większości przypadków na nieznaczną poprawę wyników. Największą różnicę zauważamy w przypadku metody k-najlbliższych sąsiadów oraz klasyfikacji binarnej z użyciem naiwnego klasyfikatora Bayesa .

Metoda	3 klasy	2 klasy
Naiwny klasyfikator Bayesa	0.48777053838	0.696734586991
K-najbliższych sąsiadów	0.600301111341	0.726892707502
Maszyna wektorów nośnych	0.666216785387	0.784501945331
Sieć neuronowa	0.66308707475	0.785167348364

Tabela 6.10: Wyniki klasyfikacji dla wektorów ważonych metodą TFIDF

6.3.6 Klasyfikacja z użyciem wektorów uzyskanych metodą word2vec oraz doc2vec

W pierwszej kolejności sprawdzono wyniki dla wektorów w przestrzeni tysiąc-wymiarowej uzyskanej za pomocą implementacji algorytmu doc2vec z pakietu Gensim. Na wejściu algorytmu podano tablice lematów dla każdego dokumentu oraz ciąg automatycznie wygenerowanych etykiet służących do oznaczania pojedynczych komentarzy.

Metoda	3 klasy	2 klasy
Naiwny klasyfikator Bayesa	0.510192157066	0.62974660814
K-najbliższych sąsiadów	0.55450216741	0.609872306464
Maszyna wektorów nośnych	0.577058548606	0.655416250998
Sieć neuronowa	0.524741653859	0.625491071429

Tabela 6.11: Dokładność klasyfikacji z użyciem wektorów wygenerowanych algorytmem doc2vec

Jak widzimy w tabeli 6.11, użycie wektorów uzyskanych dzięki przetwarzaniu algorytmem doc2vec spowodowało spadek dokładności wszystkich klasyfikatorów. Ciekawy wydaje się przypadek sieci neuronowej, która zaliczyła największy spadek dokładności uzyskując wynik porównywalny z naiwnym klasyfikatorem Bayesa. Ostatnią spośród testowanych metod reprezentacji wektorowej uzyskano za pomocą algorytmu word2vec. W pierwszym etapie wygenerowano wektory dla lematów występujących w korpusie *Komentarze*. Wektory te zostały osadzone w przestrzeni trójwymiarowej – zwykle stosuje

się kilkaset wymiarów, jednak zastosowanie tylko trzech wymiarów wynika ze sposobu ich użycia. W nowym wektorze każdy lemat był reprezentowany przez podwektor o trzech elementach. W przypadku, gdy lemat nie występuje w komentarzu, podwektor ten składa się z samych zer. Jeżeli w komentarzu dany lemat występuje co najmniej raz, w odpowiednim miejscu zostaje wstawiony podwektor będący wektorem lematu wygenerowanym przez algorytm word2vec. Łączna długość wektora stanowi więc trzykrotność liczby unikalnych lematów występujących w korpusie *Komentarze* i wynosi 36225.

Metoda	**3 klasy**	**2 klasy**
Naiwny klasyfikator Bayesa	0.489262702096	0.698961492418
K-najbliższych sąsiadów	0.507476890166	0.574812699521
Maszyna wektorów nośnych	0.622994099696	0.746756783719
Sieć neuronowa	0.57625	0.705223713089

Tabela 6.12: Dokładność klasyfikacji dla wektorów powstałych z wykorzystaniem algorytmu word2vec

W tabeli 6.12 wyraźnie widać, iż dla większości przypadków ta metoda reprezentacji tekstu nie sprawdziła się – dokładność udało się poprawić jedynie w binarnej klasyfikacji z użyciem naiwnego klasyfikatora Bayesa.

6.4 Wykorzystanie wytrenowanych klasyfikatorów w celu cenzurowania obraźliwych komentarzy na portalu Wirtualna Polska

Eksperymenty w rozdziale 6.3 przeprowadzono w celu znalezienia optymalnej metody wykrywania wypowiedzi obraźliwych dla zgromadzonego korpusu *Komentarze*. Ostatecznie wybrana została klasyfikacja binarna (klasyfikator wytrenowany na klasach skrajnych korpusu *Komentarze*). W tej kategorii najlepszym algorytmem okazała się sieć neuronowa wytrenowana na reprezentacji tekstu w postaci wektora lematów ważonego metodą TFIDF. W celu praktycznego wykorzystania klasyfikatora stworzono napisaną w języku JavaScript wtyczkę do przeglądarki Google Chrome.[4] Zadaniem wtyczki jest ekstrahowanie komentarzy pochodzących z portalu Wirtualna Polska oraz przesyłanie ich na serwer, gdzie następuje binarna klasyfikacja komentarza. Aby móc odpowiednio obsłużyć działającą w przeglądarce wtyczkę, napisano w języku Python (z użyciem modułu web.py) serwer obsługujący jedną metodę POST, co pozwala na dwustronną komunikację wtyczki z działającym zdalnie klasyfikatorem. Dokładna zasada działania tego systemu przedstawia się następująco

Po stronie wtyczki:

1. Załadowanie w przeglądarce Google Chrome strony z portalu Wirtualna Polska. Wtyczka automatycznie ekstrahuje wszystkie komentarze znajdujące się na stronie.

2. Treść każdego komentarza zostaje wysłana asynchronicznie wysłana metodą POST na działający lokalnie serwer.

3. W przypadku wywołania zakończonego sukcesem, sprawdzona zostaje odpowiedź serwera. Jeżeli odpowiedź serwera to ciąg znaków obr, zamiast ocenionego komentarza na stronę zostaje wstawiony ciag $\langle strike \rangle$*treść komentarza*$\langle strike/ \rangle$, przez co w przeglądarce komentarz ten widoczny jest jako przekreślony tekst.

Po stronie serwera:

1. Odebranie zapytania wysłanego przez wtyczkę.

[4] `https://github.com/krzjoa/Cenzor`

2. Przekształcenie komentarza do multizbioru termów, przechodząc proces przetwarzania przedstawiony w podrozdziale 6.2.

3. Przekształcenie na multizbioru termów do postaci wektora w przestrzeni, w której liczba wymiarów zgodna jest z liczbą wymiarów określonych podczas trenowania klasyfikatora.

4. Przekształcenie wektora na wektor ważony metodą TFIDF.

5. Użycie wcześniej wytrenowanego klasyfikatora binarnego.

6. Wysłanie wysłanie odpowiedzi do wtyczki.

Przykładowy wynik działania wtyczki zaprezentowano na rysunku 6.13 Możemy się domyślać, że np. pierwszy komentarz wykreślony został ze względu na występowanie w nim słów takich jak „koryto" i „menele".

[5] `http://wiadomosci.wp.pl/kat,1342,title,Jaroslaw-Walesa-zaskoczyl-PO-Jakie-sa-powody-jego-decyzji,wid,18363821,wiadomosc.html`

Rysunek 6.13: Przykład działania systemu dla strony[5]

ROZDZIAŁ 7

Podsumowanie

Celem projektu zaprezentowanego w niniejszej pracy było stworzenie systemu służącego do automatycznego cenzurowania obraźliwych komentarzy z zastosowaniem uczenia maszynowego. Zgromadzono i oznaczono korpus liczący łącznie 7358 komentarzy podzielonych na 6 klas – stworzono w tym procesie – przy okazji powstały narzędzia ułatwiające pobieranie komentarzy oraz ich ręczną klasyfikację. Opracowano autorskie narzędzie *preprocessingu* tekstów w języku polskim, które w procesie przetwarzania tekstu dokonuje jego lematyzacji oraz automatycznego poprawiania. Wszystkie wymienione narzędzia oraz korpus zostały udostępnione na portalu GitHub do ewentualnego późniejszego wykorzystania w innych projektach. Przetestowano cztery metody uczenia maszynowego, których implementacje dostępne są w popularnych pakietach dla języka Python, dodatkowo zaimplementowano dwie odmiany naiwnego algorytmu Bayesa (w języku Python; projekt dostępny jest na portalu GitHub). Podczas eksperymentów sprawdzono różne rodzaje wektorowej reprezentacji tekstu. Pary ⟨algorytm uczenia maszynowego, metoda wektorowej reprezentacji tekstu⟩ zostały sprawdzone podczas dwóch rodzajów klasyfikacji: binarnej oraz trójklasowej. W każdym przypadku udało się uzyskać rezultaty lepsze od wersji *baseline*. Najlepszy klasyfikator został użyty do stworzenia systemu umożliwiającego cenzurowanie komentarzy na portalu Wirtualna Polska, działającego za pośrednictwem wtyczki do przeglądarki Google Chrome. System ten w postaci zaprezentowanej w niniejszej pracy otwiera szerokie pole do rozmaitych ulepszeń. Zdaniem autora tej pracy, udoskonaleń tych upatrywać należy przede wszystkich w poprawie przetwarzania tekstu wejściowego: lepszej korekcie błędów oraz wyciąganiu informacji z często pojawiających się neologizmów czy niestandardowo zapisywanych wulgaryzmów.

Bibliografia

[1] Agarwal, A., Xie, B., Vovsha, I., Rambow, O., Passonneau, R. (2011). Sentiment Analysis of Twitter Data. W *Workshop on Language in Social Media LSM 2011. Proceedings of the Workshop*, s. 30-38.

[2] Agarwal, S., Sureka, A. (2016). *Spider and the Flies : Focused Crawling on Tumblr to Detect Hate Promoting Communities.*

[3] Atkinson, K. (2004). 8. How Aspell Works. Z *GNU Aspell*, `http://aspell.net/0.50-doc/man-html/8_How.html`, (dostęp: 29.04.2016)

[4] Blei, D. M., Ng, A. Y., Jordan, M. I., (2003). Latent Dirichlet Allocation. W *Journal of Machine Learning Research 3 (2003)* s. 993-1022

[5] Christen, P. (2012). *Data Matching. Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.*Springer-Verlag Berlin Heidelberg.

[6] Cichosz, P. (2007). *Systemy uczące się (wydanie drugie).* Warszawa, Wydawnictwo Naukowo-Techniczne.

[7] Cunningham, P., Cord, M., i Delany, S. J. (2008). Supervised Learning. W *Machine Learning Techniques for Multimedia. Case Studies on Organization and Retrieval* s. 21-49. Springer-Verlag Berlin Heidelberg.

[8] Djuric, N., Zhou, J., Morris, R., Grbovic, M., Radosavljevic, V., i Bhamidipati, N. (2015). Hate Speech Detection with Comment Embeddings. W *WWW '15 Companion Proceedings of the 24th International Conference on World Wide Web* s. 29-30. New York, ACM.

[9] Ghahramani, Z. (2004). Unsupervised Learning. W *Advanced Lectures on Machine Learning. ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures* s. 72-112. Springer Berlin Heidelberg.

[10] Graham-Cumming, J. (2006). Does Bayesian Poisoning exist?, `www.jgc.org/blog/poison.pdf.gz`, (dostęp: 2.04.2016)

[11] Guo, J. (n.d.). *A Simple Tutorial on Theano*, `http://ir.hit.edu.cn/~jguo/docs/notes/a_simple_tutorial_on_theano.pdf`, (dostęp: 7.05.2016)

[12] Guyon, I., i Elisseeff, A. (2006). An Introduction to Feature Extraction. W *An Introduction to Feature Extraction. Foundations and Applications* s. 1-25. Springer Berlin Heidelberg.

[13] Hsu, C.-W., i Lin, C.-J. (2002). A Comparison of Methods for Multi-class Support. W *IEEE Transactions on Neural Networks* , 13 (Issue 2),s. 415 - 425.

[14] Joachims, T. (1996). *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.*

[15] Kaggle. (2014). Retrieved 05 28, 2016, from Bag of Words Meets Bags of Popcorn, `https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-3-more-fun-with-word-vectors`, (dostęp: 28.05.2016)

[16] Komiya, K., Sato, N., Fujimoto, K., i Kotani, Y. (n.d.). *Negation Naive Bayes for Categorization of Product Pages on the Web.*

[17] Krawiec, K., i Stefanowski, J. (2004). *Uczenie maszynowe i sieci neuronowe (Wydanie II)*. Poznań, Wydawnictwo Politechniki Poznańskiej.

[18] Kriesel, D. (2007). *A Brief Introduction to Neural Networks*. Z `http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf`, (dostęp: 4.04.2016)

[19] Krizhevsky, A., Sutskever, I., i Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional. W *Neural Information Processing Systems Conference*.

[20] Kulesza, T., Amershi, S., Caruana, R., Fisher, D., i Charles, D. (2014). Structured labeling for facilitating concept evolution in machine learning. W *CHI '14 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* s. 3075-3084. Nowy Jork, ACM New York.

[21] McCallum, A., i Nigam, K. (1998). *A Comparison of Event Models for Naive Bayes Text Classification.*

[22] Medhat, W., Hassan, A., i Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. W *Ain Shams Engineering Journal*, s. 1093–1113.

[23] Mikolov, T., Chen, K., Corrado, G., i Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space.*

[24] Miłkowski, M. (2007). Morfologik. Strona projektu morfologik - analizator morfologiczny + słownik morfologiczny + korektor gramatyczny + biblioteki. Retrieved 04 29, 2016, from Znaczniki Morfologika: `http://morfologik.blogspot.com/2007/09/znaczniki-morfologika.html`, (dostęp: 29.04.2016)

[25] Oakes, M. P. (2014). Naive Bayes. W *Literary Detective Work on the Computer* s. 91. John Benjamins Publishing Company.

[26] Pańczyk, M., i Smołka, J. (2014). Wykorzystanie algorytmów przetwarzania języka naturalnego do monitorowania treści mediów internetowych. W *LOGISTYKA - 2014, nr 6*, s.8321-8330.

[27] Peng, F., i Schuurmans, D. (2003). *Combining Naive Bayes and n-Gram Language Models for Text Classification.*

[28] Razavi, A. H., Inkpen, D., Uritsky, S., i Matwin, S. (2010). Offensive Language Detection Using Multi-level Classification. Advances in Artificial Intelligence.W *23rd Canadian Conference on Artificial Intelligence, Canadian AI 2010, Ottawa, Canada, May 31 – June 2, 2010. Proceedings*, s. 16-27. Ottawa, Springer Berlin Heidelberg.

[29] Rennie, J. D., Shih, L., Teevan, J., i Karger, D. R. (2003). *Tackling the Poor Assumptions of Naive Bayes Text Classifiers.*

[30] Shi, G., Ying, Y., i Yin, Y. (2013). *The reCaptcha Helper: a Machine Learning Study.*

[31] Somanchi, S. H. (2015). The mail you want, not the spam you don't. Z *Official Gmail Blog*, `https://gmail.googleblog.com/2015/07/the-mail-you-want-not-spam-you-dont.html`, (dostęp: 2.04.2016)

[32] Sprengers, M. (2009). *The Effects of Different Bayesian Poison Methods on the Quality.*

[33] Stern, H. M. (2004). A Linguistics-Based Attack on Personalised Statistical E-mail Classifiers ,W *Technical Report CS-2004-06. 1st Conference on E-mail and AntiSpam (CEAS) Dalhousie University.*

[34] Stokowiec, W. (2015). word2vec dla Polskiego Internetu. Z *Shallow Learning. Natural Language Processing Laboratory*, `http://opi-lil.github.io/assets/other/2015_11_20_Meetup_word2vec.pdf`, (dostęp: 28.05.2016)

[35] Stone, T. (2003). *Parameterization of Naive Bayes for Spam.*

[36] Szałkiewicz, Ł. (2013). Lematyzacja w ręcznej anotacji milionowego podkorpusu Narodowego Korpusu Języka Polskiego - ciekawe przypadki. W *Polonica tom XXXIII*, s.133-156.

[37] Tomanek, K. (2014). Analiza sentymentu – metoda analizy danych jakościowych. Przykład zastosowania oraz ewaluacja słownika RID i metody klasyfikacji Bayesa w analizie danych jakościowych. W *Przegląd Socjologii Jakościowej* , s. 118-136.

[38] Varol, C., Bayrak, C., i Wagner, R. (2010). Application of the Near Miss Strategy and Edit Distance to Handle Dirty Data. W *Data Engineering. Mining, Information and Intelligence* s. 91-101. Springer US.

[39] Weiss, S. M., Indurkhya, N., i Zhang, T. (2010). From Textual Information to Numerical Vectors. W *Fundamentals of Predictive Text Mining* s. 13-38. Springer-Verlag London.

[40] Wikipedia. (n.d.). *Wikipedia:Stopwords*, `https://pl.wikipedia.org/wiki/Wikipedia:Stopwords`, (dostęp: 15.05.2016)

[41] Wikipedia (n.d.), *Damerau–Levenshtein distance*, `https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance`, (dostęp: 25.05.2016)

[42] Wittel, G. L., i Wu, S. F. (2004). On attacking statistical spam filters. W *First Conference on Email and Anti-Spam (CEAS)*. Mountain View.

[43] Xiang, G., Fan, B., Wan, L., Hong, g. J., i Rose, C. P. (2012). Detecting offensive tweets via topical feature discovery over a large scale twitter corpus. W *CIKM '12 Proceedings of the 21st ACM international conference on Information and knowledge management* s. 1980-1984. New York, ACM.

[44] Xu, Z., i Zhu, S. (2010). *Filtering Offensive Language in Online Communities using Grammatical Relations.*

[45] Zhao, J., i Yun, Y. (2009). Proximity Language Model. A Language Model beyond Bag of Words through Proximity.

[46] Zhu, X., i B.Goldberg, A. (2009). *Introduction to Semi-Supervised Learning*. Morgan tutaj ampersan Claypool.