Uniwersytet im. A. Mickiewicza w Poznaniu

Wydział Matematyki i Informatyki

Praca magisterska

Przegląd metod uczenia maszynowego stosowanych w analizie wydźwięku.

An Overview of Machine Learning Methods Used in Sentiment Analysis.

Justyna Laska

Numer albumu: 407254

Kierunek: Informatyka, Specjalność: Systemy inteligentne

Promotor:

prof. UAM dr hab. Krzysztof Jassem

Poznań, 2019

Poznań, dnia .....

Oświadczenie

Ja, niżej podpisana Justyna Laska studentka Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt: *Przegląd metod uczenia maszynowego stosowanych w analizie wydźwięku* napisałam samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałam z pomocy innych osób, a w szczególności nie zlecałam opracowania rozprawy lub jej części innym osobom, ani nie odpisywałam 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

[ ]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....

(czytelny podpis studenta)

Streszczenie

Głównym celem pracy magisterskiej jest przegląd metod uczenia maszynowego stosowanych w analizie wydźwięku. Opisano najważniejsze informacje na temat uczenia maszynowego oraz analizy wydźwięku. Stworzono projekt magisterski, który automatycznie rozpoznaje wydźwięk tekstu dostarczonego przez użytkownika, korzystając z metod uczenia maszynowego. W pracy przedstawiono główny cel projektu, wykorzystane w nim narzędzia, opis etapów analizy wydźwięku tekstów w serwisie społecznościowym *Twitter*, przeprowadzone eksperymenty i ich wyniki.

Słowa kluczowe: przetwarzanie języka naturalnego, klasyfikacja tekstu, analiza wydźwięku, uczenie maszynowe

Abstract

The main aim of the master's thesis is the exploration of the machine learning methods used in sentiment analysis. The thesis introduces basic information on the topics of machine learning and sentiment analysis. The project designed for the sake of the thesis automatically recognises the sentiment of the text provided by the user, using the machine learning methods. This paper presents the main project objective, the description of the tools used in the project, the step-by-step procedure of sentiment analysis of texts appearing at the *Twitter* service, conducted experiments and their results.

Key words: natural language processing, text classification, sentiment analysis, machine learning

Spis treści

Wstęp

Liczba dostępnych informacji na stronach internetowych stale rośnie. Teksty wyrażające opinie są dostępne na stronach z recenzjami, forach, blogach i mediach społecznościowych. Za pomocą analizy wydźwięku możliwe jest uzyskanie opinii ludzi na dany temat. Dostępność dużych wolumenów tekstu do trenowania modeli jest ważnym czynnikiem przyczyniającym się do wykorzystania metod uczenia maszynowego w klasyfikacji tekstów pod kątem wydźwięku.

Celem niniejszej pracy magisterskiej jest przegląd metod uczenia maszynowego stosowanych w analizie wydźwięku. Rozprawa składa się z pięciu rozdziałów. Pierwszy wyjaśnia najważniejsze informacje dotyczące analizy wydźwięku. Opisuje jej zastosowanie, poziomy oraz etapy. Kolejny rozdział prezentuje wiedzę teoretyczną związaną z uczeniem maszynowym. Omówione są rodzaje uczenia: nadzorowane, nienadzorowane oraz przez wzmacnianie wraz z wybranymi algorytmami. Przybliżono, czym jest ewaluacja modelu. Trzeci rozdział ukazuje krótki opis artykułu, którego celem była analizy wydźwięku recenzji filmów za pomocą metod uczenia maszynowego. Rozdział czwarty poświęcony jest wykorzystanym w projekcie narzędziom. Natomiast piąty rozdział przedstawia główny cel i opis projektu magisterskiego. W skład projektu wchodziły następujące etapy automatycznej analizy wydźwięku *tweet'ów*: pozyskanie oraz przygotowanie zbioru danych, ekstrakcje cech, uczenie klasyfikatorów wykorzystujących różne algorytmy: naiwnego algorytmu Bayes'a, regresji logistycznej, k-najbliższych sąsiadów, k-średnich, drzew decyzyjnych, maszyny wektorów nośnych oraz sztucznych sieci neuronowych. W rozdziale opisano również przeprowadzone eksperymenty oraz ich wyniki.

Rozdział 1

Wprowadzenie do analizy wydźwięku

Niniejszy rozdział poświęcony jest omówieniu pojęcia analizy wydźwięku. Są tu opisane zastosowanie, poziomy oraz etapy takiego procesu.

1.1. Definicja pojęcia analizy wydźwięku

Analiza wydźwięku[1] to proces badania opinii, postaw, emocji wyrażonych w tekście w celu określenia nastawienia autora do danego bytu (np. czy jest on pozytywny, negatywny czy neutralny).

1.2. Zastosowania

W erze internetu łatwo zebrać opinie od osób z całego świata. Ludzie chcą przeglądać handlowe strony internetowe (np. Amazon, eBay), witryny z opiniami online (np. TripAdvisor, Opineo, Yelp) lub media społecznościowe (np. Facebook, Twitter) i uzyskać informację zwrotną na temat tego, jak określony produkt lub usługa postrzegane są na rynku.

Analiza wydźwięku jest przydatna w monitorowaniu tekstów zamieszczanych przez internautów. Dzięki niej możliwe jest uzyskanie wiedzy o opinii publicznej na

[1] ang. *Sentiment analysis, opinion mining, emotion AI*

konkretny temat, np. czy recenzja na temat nowego filmu jest pozytywna, czy negatywna lub co ludzie myślą np. o nowy smartfonie. Współpracownicy prezydenta Obamy wykorzystali analizę nastrojów, aby ocenić opinię publiczną na temat ogłoszeń politycznych i komunikatów kampanii przed wyborami prezydenckimi w 2012 roku [9]. Natomiast inni badacze wykazali, że zmiany nastrojów w mediach społecznościowych korelują ze zmianami na giełdzie [3][18].

1.3. Analiza wielopoziomowa

Analiza wydźwięku może występować na różnych poziomach: na poziomie dokumentu, na poziomie zdania lub na poziomie aspektu / cechy.

1.3.1. Poziom dokumentu[2]

Wydźwięk określany jest na podstawie całego dokumentu. Ten poziom analizy zakłada, że dokumenty wyrażają opinię na temat poszczególnego bytu (np. pojedynczego produktu). Dlatego nie ma zastosowania do dokumentów, które oceniają lub porównują wiele jednostek. [13]

1.3.2. Poziom zdania[3]

Analiza na tym poziomie korzysta tylko z pojedynczych zdań. Określa, czy poszczególne zdania zawierają opinię pozytywną, negatywną lub neutralną (brak opinii).[13]

1.3.3. Poziom aspektu / cechy[4]

Klasyfikacja wydźwięku na poziomie dokumentu i zdania nie zawiera szczegółów na temat postrzegania konkretnej cechy w ocenianej jednostce. Nie odkrywa także celów takiej recenzji. Poziom aspektu umożliwia dokładniejszą analizę.

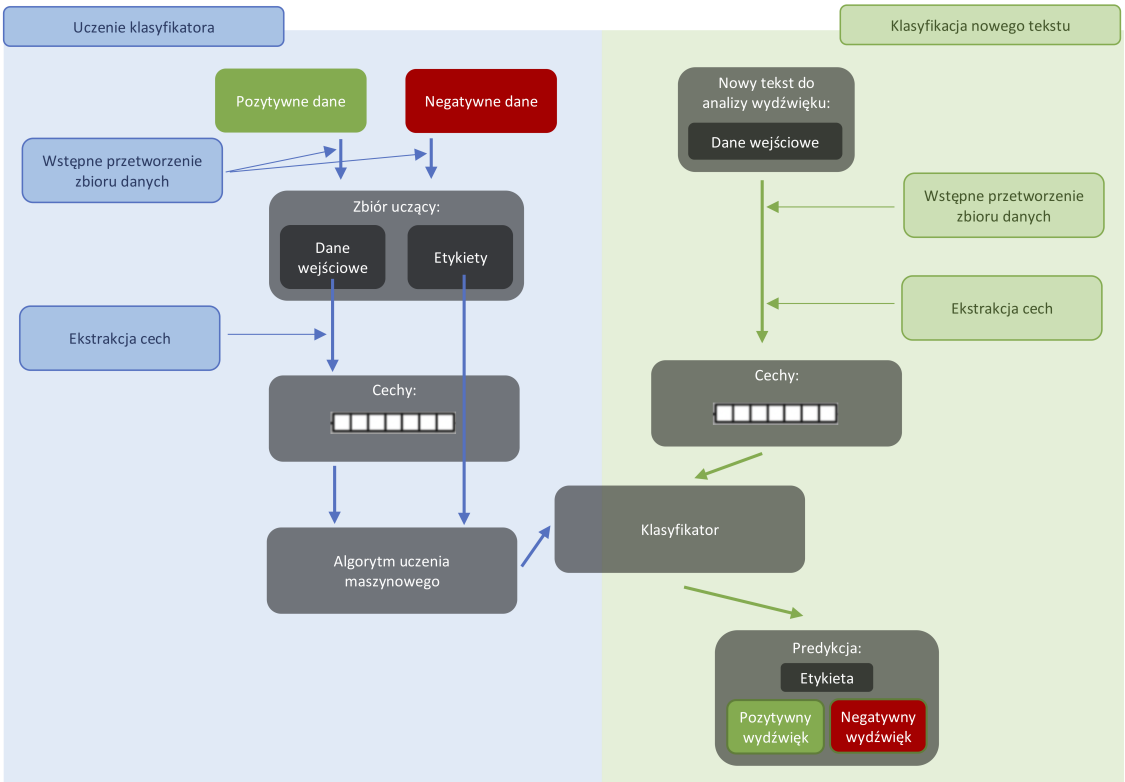
[2] ang. *Document level*
[3] ang. *Sentence level*
[4] ang. *Aspect / feature level*

Na przykład zdanie *"Wymiary iPhone'a są idealne, ale jego jakość dźwięku jest kiepska"* ocenia dwa aspekty, wymiary i jakość dźwięku telefonu (obiektu). Opiniodawcy mogą wydawać różne oceny na temat różnych cech tego samego podmiotu. W tym przypadku wydźwięk dotyczący wymiarów iPhone'a jest pozytywny, natomiast ten odnoszący się do jakości dźwięku – negatywny.

Wyróżnia się dwa typy opinii: opinie regularne (np. *"Coca cola smakuje rewelacyjnie"*) oraz porównawcze (np. *"Coca cola jest znacznie lepsza od Pepsi"*). Na poziomie aspektu można porównać wiele bytów w oparciu o niektóre z ich wspólnych aspektów (np. ww. wymienione napoje w oparciu o ich smak lub kolor). [13]

1.4. Architektura systemów analizy wydźwięku

Rysunek 1.1 obrazuje fazy analizy wydźwięku wykorzystującej metody statystyczne.



Rys. 1.1. Analiza wydźwięku – model ogólny.

Celem analizy wydźwięku jest automatyczne sklasyfikowanie tekstu jako pozytywnego lub negatywnego (czasem także neutralnego). Poniżej przedstawiono główne elementy takiej analizy.

1.4.1. Zbiór danych[5]

W dzisiejszych czasach istnieje wiele zasobów internetowych, w których użytkownicy mogą wyrażać swoje opinie. Istnieją dwa główne sposoby zbierania danych z internetowych zasobów: za pomocą interfejsu API[6] oraz za pomocą ekstrakcji danych internetowych. Ten krok jest bardzo ważny, ponieważ ilość i jakość gromadzonych danych bezpośrednio decyduje o tym, jak dokładny będzie klasyfikator. Na rysunku 1.1 zbiór danych składa się z pozytywnych i negatywnych danych.

1.4.2. Wstępne przetwarzanie zbiorów danych[7]

Dane gromadzone są z przeróżnych źródeł. Na przykład serwis społecznościowy *Twitter* zawiera wiele opinii na temat danych, które są wyrażane na różne sposoby przez różnych użytkowników. Takie zbiory wymagają oczyszczenia, jeśli chce się nich skorzystać w celu stworzenia modelu uczenia maszynowego.

Wstępne przetwarzanie obejmuje m.in.:

- usunięcie :
  - adresów URL (np. www.fb.com), tagów (np. #tree), odnośników (np. @user123),
  - powtarzających się znaków,
  - wszystkich znaków interpunkcyjnych, symboli, licz b,
  - wyrazów pomijanych (ignorowanych; *stop words*, np. ang.: a, about, be),
- zamienienie emotikon na ich wydźwięk (np. ☺– smile, ☹– sadness, ☼– sun, *:-¡* – sadness, *8-)* – *smile*),
- rozwinięcie akronimów (skrótów) (np. ang.: CU – See you, TQ – Thank You),

[5]ang. *The datasets*
[6]ang. *Application Programming Interface)*
[7]ang. *Pre-processing of the datasets*

1.4.3. Ekstrakcja cech[8]

Cechy to mierzalne właściwości obiektu, który jest analizowany. Są one danymi wejściowymi dla modelu[9] powstałego w skutek wytrenowania algorytmu uczenia maszynowego. Wstępnie przetworzony zestaw danych ma wiele charakterystycznych właściwości. Metoda ekstrakcji cech w przypadku przetwarzania tekstów polega na wyodrębnieniu słów kluczowych podlegających analizie z przetworzonego zestawu danych, a następnie przekonwertowaniu danych wejściowych do wektora cech.

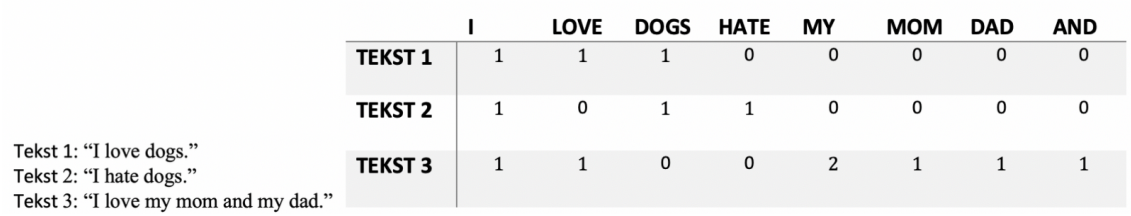
Dzięki temu możemy uzyskać, np. informacje o:

- obecności oraz częstości występowania słowa,
- pozycji termu w tekście,
- częściach mowy,
- wyrażeniach językowych (idiomach),
- kolokacjach (sens całości wynika ze znaczeń poszczególnych wyrazów),
- negacjach.

Model "worka słów"[10] pozwala nam przedstawiać tekst jako numeryczne wektory cech[11] (słów). W takim wektorze zliczona zostaje jedynie liczba wystąpień takiego słowa w danym tekście. Na rysunku 1.2 przedstawione są trzy przykładowe teksty oraz ich graficzne reprezentacje jako wektor cech o długości równej wielkości słownika.

Zazwyczaj słowa występujące w wielu przykładach zawierają mało istotne informacje. Technikę $tf - idf$[12] można użyć do zmniejszenia wagi często pojawiających

[8] ang. *Feature extraction*
[9] model – „specyfikacja zależności matematycznych zachodzących pomiędzy różnymi zmiennymi." [7]
[10] ang. *Bag-of-words*
[11] ang. *Feature vectors*
[12] ang. *Term frequency–inverse document frequency*



	I	LOVE	DOGS	HATE	MY	MOM	DAD	AND
TEKST 1	1	1	1	0	0	0	0	0
TEKST 2	1	0	1	1	0	0	0	0
TEKST 3	1	1	0	0	2	1	1	1

Rys. 1.2. Przykładowa graficzna reprezentacja wektorów cech tekstów za pomocą modelu "worka słów".

się słów w wektorach cech. $Tf-idf$ można zdefiniować jako TF[13] – iloczyn częstości wyrazu (frekwencja wyrazu) i IDF[14]– odwrotnej częstości dokumentu:

$$tf-idf(t,d) = tf \times idf(t,d).$$

Dokumentem określany jest tekst, który ma zostać zbadany. Natomiast korpus to zbiór dokumentów, do którego chcemy porównać badany dokument.

"TF pozwala określić częstotliwość występowania danej frazy w konkretnym dokumencie. Jego zakres obejmuje wyłącznie jeden dokument, który ma być badany. Wartość ta jest wprost proporcjonalna do częstotliwości występowania wyrazu. Aby obliczyć TF, należy podzielić liczbę wystąpień frazy w dokumencie przez liczbę znajdujących się w nim wszystkich słów."[6] Przedstawia to poniższy wzór:

$$tf(t,d) = \frac{n_{t,d}}{n_{k,d}},$$

gdzie $tf(t,d)$ oznacza częstość słowa t w dokumencie d, $n_{t,d}$ – liczbę wystąpień słowa t w dokumencie d, natomiast $n_{k,d}$ – liczbę wszystkich słów w dokumencie d[6].

IDF pozwala sprawdzić, jak często dany term występuje we wszystkich dokumentach badanego korpusu językowego. Im częściej dany wyraz pojawia się w korpusie, tym wynik IDF będzie niższy.[6] Wartość tę można obliczyć z poniższego wzoru:

$$idf(t,d) = log\frac{d_t}{m_t},$$

gdzie $idf(t,d)$ oznacza odwrotną częstość słowa t we wszystkich dokumentach kor-

[13]ang. *Term frequency*
[14]ang. *Inverse document frequency*

pusu d, d_t – ogólną liczba dokumentów w badanym korpusie językowym, m_t – liczbę dokumentów, które zawierają co najmniej jedno wystąpienie słowa t[6].

Przykład (Patrz Rys. 1.3). Powiedzmy, że chcemy obliczyć częstotliwość frazy „love", występującej 2 razy w dokumencie, który zawiera 7 słów (Patrz *"Tekst 3"* Rys. 1.3). Przyjmijmy, że w badanym korpusie, który zawiera 3 dokumenty, wyrażenie „love" pojawia się w 2 dokumentach.

$n_{t,d}$ wynosi 2, $n_{k,d}$ – 7. Zatem $TF = \frac{2}{7}$.
d_t wynosi 3, m_t – 2. Zatem $IDF = log(\frac{3}{2})$.
Zatem dla frazy „love" $TD - IDF = \frac{2}{7} \times log(\frac{3}{2}) = 0,025$.[6]

Tekst 1: "I love dogs."
Tekst 2: "I hate dogs."
Tekst 3: "I love my mom and my dad."

	I	LOVE	DOGS	HATE	MY	MOM	DAD	AND
TEKST 1	$\frac{1}{3} \times \log\frac{3}{3} = 0$	$\frac{1}{3} \times \log\frac{3}{2} = 0,059$	$\frac{1}{3} \times \log\frac{3}{2} = 0,059$	0	0	0	0	0
TEKST 2	$\frac{1}{3} \times \log\frac{3}{3} = 0$	0	$\frac{1}{3} \times \log\frac{3}{2} = 0,059$	$\frac{1}{3} \times \log\frac{3}{1} = 0,159$	0	0	0	0
TEKST 3	$\frac{1}{7} \times \log\frac{3}{3} = 0$	$\frac{1}{7} \times \log\frac{3}{2} = 0,025$	0	0	$\frac{2}{7} \times \log\frac{3}{1} = 0,136$	$\frac{1}{7} \times \log\frac{3}{1} = 0,068$	$\frac{1}{7} \times \log\frac{3}{1} = 0,068$	$\frac{1}{7} \times \log\frac{3}{1} = 0,068$

Rys. 1.3. Przykładowa graficzna reprezentacja wektorów cech tekstów za pomocą metody *Tfidf*.

1.4.4. Uczenie / trenowanie[15]

Uczenie klasyfikatora umożliwia przyszłe przewidywania nieznanych danych. Podczas trenowania model uczy się kojarzyć określone dane wejściowe (np. tekst) z odpowiednimi danymi wyjściowymi (etykietami) w oparciu o zbiór testowy użyty do trenowania. Wyróżniamy dwa typy automatycznej analizy wydźwięku: maszynowe (stosujące algorytmy uczenia maszynowego – Patrz Rozdział 2) oraz semantyczne (oparte na słownikach lub korpusie).

1.4.5. Klasyfikacja[16]

Tekst, który chcemy sklasyfikować poddany zostaje ekstrakcji cech w ten sam sposób, jak w przypadku danych uczących. Po wytrenowaniu modelu klasyfikatora moż-

[15] ang. *Training*
[16] ang. *Classification*

liwe jest wprowadzenie nowego tekstu (w postaci wektora cech) do modelu, który wygeneruje przewidywany wydźwięk (etykietę).

1.5. Podsumowanie

Analiza wydźwięku jest cennym procesem, ponieważ liczba publicznie i prywatnie dostępnych informacji na stornach internetowych ciągle idzie w górę. Za pomocą automatycznych systemów analizy wydźwięku opartych na technikach uczenia maszynowego te nieuporządkowane dane mogą w zautomatyzowany sposób zostać przekształcone w ustrukturyzowane dane opinii publicznej na temat produktów, usług, marek, polityki lub dowolnego tematu, na który ludzie mogą wyrażać opinie. Analiza może występować na poziomie dokumentu, zdania lub cechy. Pierwszym etapem przeprowadzenia analizy wydźwięku jest przetworzenie zgromadzonego korpusu. Taki zestaw danych oraz algorytm uczenia maszynowego są wykorzystywane w etapie uczenia się, którego efektem końcowym jest wytrenowany klasyfikator. Ostatnim krokiem jest przetestowanie wytrenowanego klasyfikatora na innych danych (ewaluacja modelu).

Rozdział 2

Uczenie maszynowe[1]

W tym rozdziale zostaną opisane pojęcia dotyczące uczenia maszynowego oraz jego rodzaje wraz z przykładowymi algorytmami. Przybliżony będzie również proces ewaluacji modelu.

2.1. Wprowadzenie

W tradycyjnym systemie programowania inżynierowie oprogramowania wykorzystują swoją pomysłowość do opracowania rozwiązania (najczęściej zbioru reguł generowania danych wyjściowych) i sformułowania go jako precyzyjnego **programu**, który komputer może wykonać na **podanych danych** w celu wygenerowania **pożądanego wyniku** (Patrz Rys. 2.1).

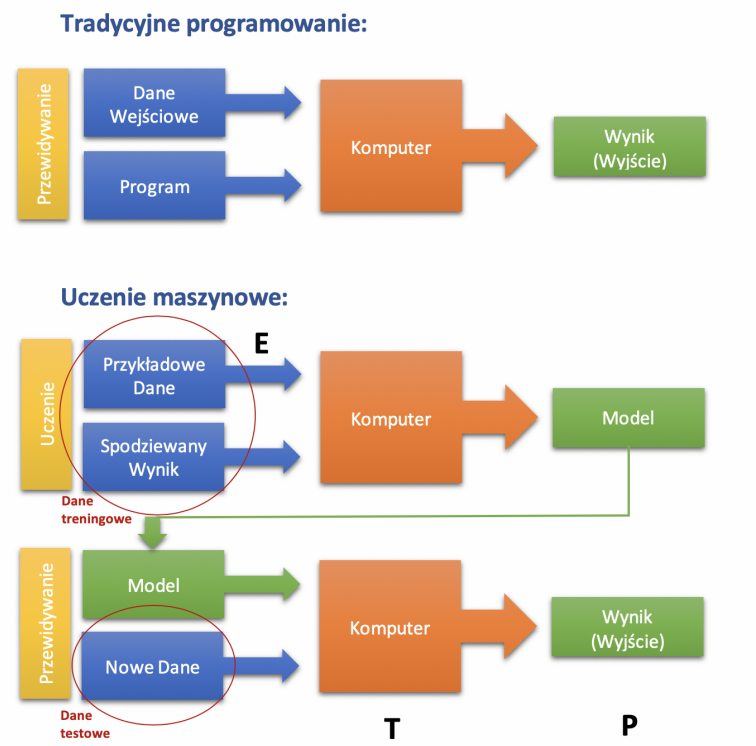
Według definicji w encyklopedii PWN *uczenie się* to „modyfikacja zachowania się jednostki w wyniku jej dotychczasowych doświadczeń"[24]. A zatem czym jest uczenie maszynowe? Istnieją różne definicje. Jedna z nich mówi, że:

> „Program komputerowy (maszyna) uczy się na podstawie doświadczenia E w odniesieniu do pewnej klasy zadań T i miary efektywności P, jeśli jego efektywność wykonywania zadania T (mierzona za pomocą P) poprawia się wraz z doświadczeniem E"[2]

[1] ang. *Machine learning*

[2] oryg.: *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

— Douglas Adams[14]



Rys. 2.1. Graficzne przedstawienie kontrastu pomiędzy tradycyjnym programowaniem a uczeniem maszynowym (tu: nadzorowanym).

W uczeniu maszynowym dostarczane są **dane wejściowe** (zmienna objaśniająca, wejściowa) wraz ze **spodziewanymi wynikami** (zmienna objaśniana, wyjściowa (celu)), (Patrz E – Rys. 2.1). Rolą oprogramowania jest poznanie zasad, które generują obserwowane rezultaty poszczególnych danych wejściowych (uczenie nadzorowane). System trenuje **model**, który reprezentuje to, czego do tej pory nauczył się system z danych treningowych (Patrz Rys. 2.1).

Po tej fazie uczenia, przeszkolone oprogramowanie jest gotowe do wnioskowania. Biorąc pod uwagę **nowy element** danych wejściowych, **wyszkolony model** może teraz wywnioskować **oczekiwany wynik** (Patrz T – Rys. 2.1). Zestaw danych testowych, przewidzianych oraz oczekiwanych wyników określa efektywność przewidywania wyniku (Patrz P – Rys. 2.1).

Podsumowując, aby mieć dobrze zdefiniowany problem uczenia, trzeba zidentyfikować trzy cechy: klasę zadań T, miarę wydajności do poprawy P oraz źródło doświadczenia E.

Jednym z zastosowań uczenia maszynowego jest rozpoznanie wydźwięku tekstów. W zadaniu tym składowe uczenia maszynowego można zdefiniować następująco:

- Zadanie T – rozpoznawanie i klasyfikowanie wydźwięku tekstu.
- Miara wydajności P – procent poprawnie sklasyfikowanych tekstów.
- Doświadczenie szkoleniowe E – baza danych tekstów oraz ich poprawnie określony wydźwięk.

2.2. Rodzaje uczenia

Istnieją trzy najczęściej wymieniane kategorie algorytmów uczenia maszynowego: nadzorowane, nienadzorowane oraz przez wzmacnianie. Poniżej umieszczony rysunek 2.2 przedstawia je wraz z metodami.

2.2.1. Uczenie nadzorowane[3]

W uczeniu nadzorowanym algorytmy tworzą funkcję, która przewiduje przyszły wynik w oparciu o dane wejściowe. Modele predykcyjne otrzymują instrukcje dotyczące tego, czego należy się nauczyć i jak się tego nauczyć.

Zbiór danych uczących, który dostarczany jest maszynie, składa się ze zbioru cech/atrybutów wejściowych (wektor $\vec{x}$) oraz z poprawnie zaetykietowanej odpowiedzi – wartości wyjściowej ($f(\vec{x})$). System uczy się modelu (funkcji h), który ma jak najlepiej aproksymować[4] funkcję f w celu poprawnej predykcji etykiet przykładów, z którymi system jeszcze się nie zetknął (Patrz Rys 2.1). Głównymi zagadnieniami w uczeniu nadzorowanym są klasyfikacja i regresja. Pierwsza służy do przewidywania dyskretnych odpowiedzi, natomiast druga – liczb rzeczywistych.

[3] ang. *Supervised Learning*

[4] Aproksymacja – procedura zastępowania jednej funkcji (funkcja aproksymowana) inną funkcją (funkcja aproksymująca) w taki sposób, aby funkcje te niewiele się różniły w sensie określonej normy.[20]

Rys. 2.2. Rodzaje algorytmów uczenia maszynowego.

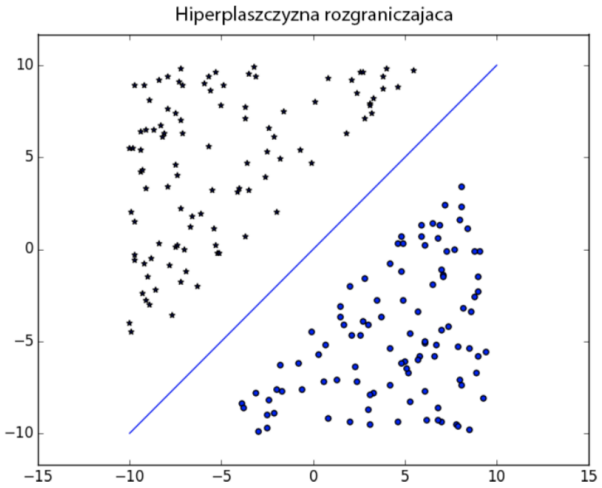
2.2.1.1. Maszyna wektorów nośnych (SVM[5])

SVM to nadzorowany algorytm uczenia maszynowego, który można stosować zarówno w wyzwaniach klasyfikacji, jak i regresji. Jest jednak najczęściej używany w problemach z klasyfikacją. Wysoka wymiarowość przestrzeni cech podnosi zarówno złożoność próby, jak i złożoność obliczeniową.

Wektory nośne[6] to współrzędne indywidualnej obserwacji, a maszyna wektorów nośnych to granica, która najlepiej rozdziela dwie klasy (hiperpłaszczyzna[7]). Usunięcie wektorów nośnych zmieni położenie hiperpłaszczyzny. Na rysunku 2.3 przedstawiona została przykładowa hiperpłaszczyzna rozgraniczająca klasy.

[5]ang. *Support Vector Machine*
[6]ang. *Support Vectors*
[7]ang. *Hyper-plane/ line*

Celem algorytmu SVM jest szukanie hiperpłaszczyzny, która „najlepiej" rozdziela klasy treningowego zbioru danych, czyli takiej, która maksymalizujące odległość do najbliższego punktu w każdej z klas. Ograniczenie algorytmu do generowania hiperpłaszczyzny o dużym marginesie może przynieść małą złożoność próbki, nawet jeśli wymiar przestrzeni cech jest wysoki (a nawet nieskończony). Znalezienie takiej hiperpłaszczyzny to problem optymalizacyjny[7][23].



Rys. 2.3. Hiperpłaszczyzna rozgraniczająca klasy.

Źródło: [7].

2.2.1.2. Naiwny klasyfikator bayesowski[8]

Jednym z najczęściej stosowanych metod uczenia maszynowego jest klasyfikator oparty na twierdzeniu Bayes'a. Zakłada się wzajemną niezależność zmiennych niezależnych (założenie te mocno upraszcza model, stąd nazwa: "klasyfikator naiwny"). Występują różne wersje naiwnego klasyfikatora Bayes'a w zależności od zastosowanego rozkładu prawdopodobieństwa[4].

Załóżmy, że istnieje zbiór d atrybutów $x_1, x_2, \cdots, x_d$, oraz atrybut decyzyjny y mogący przyjmować wartości ze zbioru $y_1, y_2, \cdots, y_k$ (etykiety, np. 0 i 1). Chcąc określić decyzję dla przykładu opisanego wartościami atrybutów $(v_1, v_2, \cdots, v_d)$ trzeba obliczyć dla każdej z klas decyzyjnych y_i prawdopodobieństwo za pomocą wzoru [21]:

$$P(y = y_i | x_1 = v_1, x_2 = v_2, ..., x_d = v_d).$$

[8] ang. *Naive Bayes*

Z twierdzenia Bayes'a można to przekształcić jako:

$$\frac{P(x_1 = v_1, x_2 = v_2, \cdots, x_d = v_d | y = y_i) P(y = y_i)}{P(x_1 = v_1, x_2 = v_2, \cdots, x_d = v_d)}.$$

$P(X|Y)$ oznacza prawdopodobieństwo a-posteriori, że przykład X należy do klasy Y, $P(Y)$ – prawdopodobieństwo a-priori wystąpienia klasy Y (tj. prawdopodobieństwo, że dowolny przykład należy do klasy Y), $P(X)$ – prawdopodobieństwo a-priori wystąpienia przykładu X[21].

Łatwo zaobserwować, że dla każdej klasy y_i mianownik jest stały, więc na potrzeby jedynie wybrania najlepszej klasy wystarczy obliczyć licznik i wybrać największą wartość. Przyjmuje się (naiwne) założenie, że dla danej etykiety cechy są od siebie niezależne[21]. Wtedy:

$$P(x_1 = v_1, x_2 = v_2, \cdots, x_d = v_d | y = y_i) = \prod_{l=1}^{d} P(x_l = v_l | y = y_i),$$

gdzie $P(x_l = v_l | y = y_i)$ wynosi $\frac{\text{liczba przypadków z } x_l = v_l \text{ w klasie } y_i}{\text{liczba wszystkich przypadków uczących w } y_i}$.

Ostatecznie uzyskuje się wzór:

$$P(y = y_i | x_1 = v_1, x_2 = v_2, \cdots, x_d = v_d) = P(y = y_i) \prod_{l=1}^{d} P(x_l = v_l | y = y_i).$$

Problemem jest oszacowanie prawdopodobieństwa wystąpienia cechy, jeśli nie pojawiła się w danych treningowych – prawdopodobieństwo to wynosi zero. Aby wyeliminować zera, stosujemy wygładzanie Laplace'a (plus 1[9]), które dodaje jeden do każdego zliczenia.

Decyzja o przynależności przykładu do klasy następuje poprzez przypisanie go do klasy z maksymalnym prawdopodobieństwem *a posteriori*[4]:

$$h_{Bayes}(x) = \underset{y \in \{y_1, y_2, \cdots, y_k\}}{\operatorname{argmax}} P(y = y_i) \prod_{l=1}^{d} P(x_l = v_l | y = y_i).$$

[9] ang. *Add-one*

2.2.1.3. Algorytm k najbliższych sąsiadów (k-nn)[10]

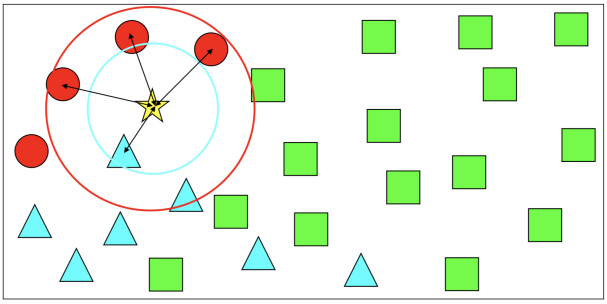
"Metoda k-najbliższych sąsiadów należy do grupy algorytmów leniwych, czyli takich, które nie tworzą wewnętrznej reprezentacji danych uczących, lecz szukają rozwiązania dopiero w momencie pojawienia się wzorca testującego, np. do klasyfikacji. Przechowuje wszystkie wzorce uczące, względem których wyznacza odległość wzorca testowego".[8]

Każdej danej należy przypisać pewien zestaw n cechujących ją wartości, a następnie umieścić ją w n-wymiarowej przestrzeni. Chcąc przyporządkować daną do już istniejącej grupy, należy znaleźć k najbliższych obiektów w przestrzeni n-wymiarowej (najbliższych dla wybranej metryki (np. Euklidesowej)), a następnie wybrać grupę najbardziej liczną. Wzór na odległość Euklidesa:

$$\|\mathbf{x} - x^i\| = \sum_{j=1}^{n} (x_j - x_j^i)^2,$$

gdzie "x^i jest zbiorem parametrów $x^i = x_1^i, ..., x_1^n$ definiujących obiekty (zwykle w postaci wektora lub macierzy danych), zaś y^i jest wartością przewidywaną, indeksem lub nazwa klasy, do której obiekt x^i należy i którą razem z innymi obiektami tej klasy definiuje"[8].

Rysunek 2.4 przedstawia w graficzny sposób przykład klasyfikacji *gwiazdki* za pomocą metody *k-nn*. Obiekt zostanie sklasyfikowany do klasy, która będzie najbliżej (trójkąt, kwadrat lub koło).



Rys. 2.4. Metoda k-najbliższych sąsiadów.

Źródło: `http://home.agh.edu.pl/~horzyk/lectures/miw/KNN.pdf`

[10] ang. *k nearest neighbours*

2.2.1.4. Regresja liniowa

Regresja liniowa jest metodą modelowania liniowej zależności między jedną lub większą liczbą zmiennych niezależnych x (zmienna objaśniająca, wejściowa) a pojedynczą zmienną zależną y (zmienna objaśniana, wyjściowa). Zarówno zmienne objaśniane i objaśniające mogą być wielkościami skalarnymi lub wektorami.

Gdy występuje pojedyncza zmienna wejściowa x, metodę określa się jako prostą regresję liniową[11]. Natomiast, gdy zawiera wiele – metodę nazywana jest wielokrotną regresją liniową[12][5].

Algorytm nosi nazwę prostej regresji liniowej[13], gdy istnieje pojedynczy atrybut wejściowy x. Natomiast dla wielu atrybutów wejściowych (np. x_1, x_2 itd.) byłaby to wielokrotna regresja liniowa[14].

Dla prostej regresji liniowej, prosta regresji jest postaci:

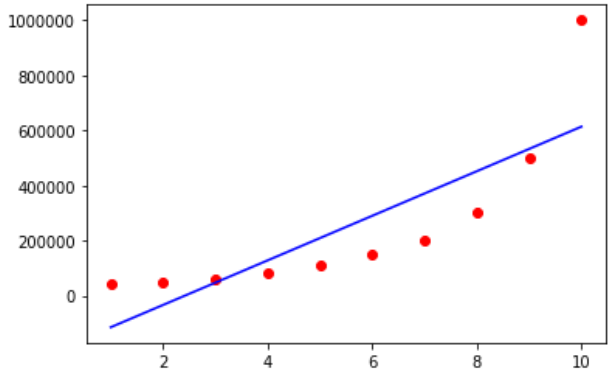
$$y = b_0 + b_1 * x,$$

gdzie współczynnik b^0 nazywany jest przecięciem, ponieważ określa, gdzie prosta regresji przecina oś y. Natomiast b^1 określane jest jako nachylenie linii regresji. W wyższych wymiarach, gdy mamy więcej niż jedno wejście x, linia nazywana jest płaszczyzną lub hiperpłaszczyzną. Postać wzoru dla wielokrotnej regresji liniowej:

$$y = b_0 + \left(b^1 * x^1\right) + \cdots + b_n x^n$$

Rysunek 2.5 przedstawia dopasowanie modelu prostej regresji liniowej. W tym przypadku modelem jest linia prosta. Analiza regresji polega jak najlepszym dopasowaniu linii do danych (punkty na wykresie).

[11] ang. *Simple Linear Regression*
[12] ang. *Multiple Linear Regression*
[13] ang. *Simple Linear Regression*
[14] ang. *Multiple Linear Regression*



Rys. 2.5. Przykładowe dopasowanie modelu prostej regresji liniowej.

Źródło: `https://animoidin.wordpress.com/2018/07/01/`

2.2.1.5. Regresja wielomianowa[15]

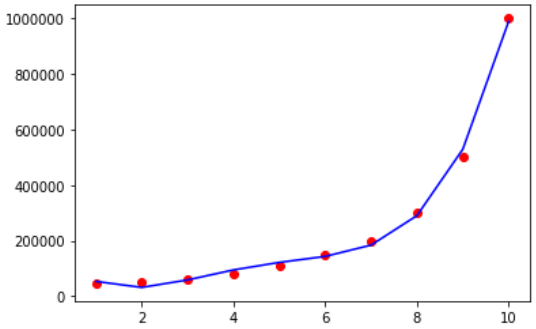
Regresja wielomianowa jest formą regresji, w której zależność między zmienną niezależną x a zmienną zależną y jest modelowana jako wielomian n-tego stopnia.

Ogólny model regresji wielomianowej:

$$y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + \cdots + b_n x^n.$$

Zakłada on, że y jest funkcją wielomianową, która chcemy przewidzieć, x jest zmienną wejściową, która znamy, a $b0, b1, b2...$ są współczynnikami regresji, które szukamy.

Rysunek 2.6 przedstawia przykładowe dopasowanie modelu regresji wielomianowej. W tym przypadku modelem jest krzywa linia (regresja krzywoliniowa). Analiza regresji polega jak najlepszym dopasowaniu linii do danych (punkty na wykresie).



Rys. 2.6. Przykładowe dopasowanie modelu regresji wielomianowej.
Źródło: `https://animoidin.wordpress.com/2018/07/01/`

[15] ang. *Polynomial Regression*

2.2.1.6. Regresja logistyczna[16]

Nazwa regresji logistycznej pochodzi od stosowanej w algorytmie funkcji logistycznej. Funkcja logistyczna, zwana także funkcją sigmoidalną, jest to krzywa w kształcie litery S, która może przyjąć dowolną liczbę o wartości rzeczywistej i odwzorować ją na wartość z zakresu od 0 do 1, ale nigdy dokładnie na tych granicach[4].

Wzór funkcji logistycznej (sigmoidalnej) ma postać:

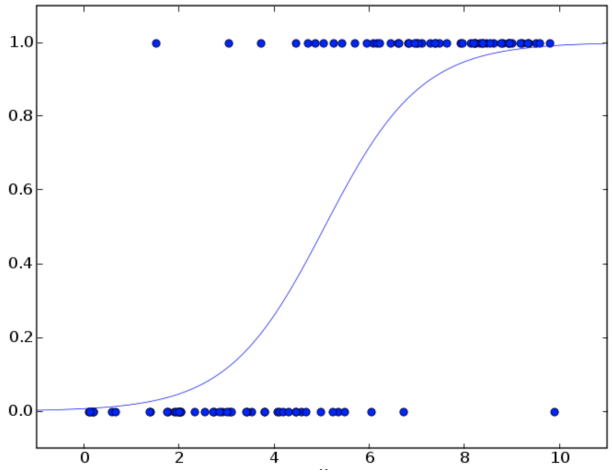
$$y(x) = \frac{1}{1 + e^{-x}},$$

gdzie e jest podstawą logarytmów naturalnych (liczba Eulera), a x jest rzeczywistą wartością liczbową, którą jest przekształcana.

Wynik regresji logistycznej, podobnie jak regresji liniowej, można przedstawić za pomocą równania. Wartości wejściowe x są łączone liniowo za pomocą wag lub wartości współczynników, aby przewidzieć wartość wyjściową y. Kluczową różnicą w stosunku do regresji liniowej jest to, że modelowana wartość wyjściowa jest wartością binarną (0 lub 1).

Zakładając, że wartość wejściowa ma postać $b^0 + b^1 * x$. Równaniem regresji logistycznej będzie $p(class = 0) = \frac{1}{1+e^{-(b^0+b^1*x)}}$. Wynikiem algorytmu jest prawdopodobieństwo wyznaczone przy użyciu funkcji logistycznej zdefiniowanej wyżej. Model regresji logistycznej pobiera dane wejściowe o wartościach rzeczywistych i dokonuje przewidywania prawdopodobieństwa wejścia należącego do klasy domyślnej (klasa 0). Jeśli prawdopodobieństwo wynosi więcej niż 0,5, możemy przyjąć wyjście jako prognozę dla klasy domyślnej (klasa 0), w przeciwnym razie przewidywanie jest dla drugiej klasy (klasa 1). Rysunek 2.7 przedstawia przykładowe dopasowanie modelu regresji logistycznej[4]. Przewiduje wartości bardzo zbliżone do 1 oraz 0.

[16] ang. *Logistic Regression*



Rys. 2.7. Przykładowe dopasowanie modelu regresji logistycznej.

Źródło: `https://st3.ning.com/topology/rest/1.0/file/get/2808358994?profile=original`

2.2.1.7. Drzewa decyzyjne[17]

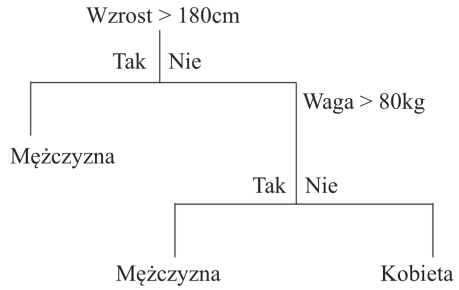
Algorytm drzewa decyzyjnego znany jest również pod bardziej nowoczesną nazwą *CART*[18], która oznacza drzewa klasyfikacji i regresji. Algorytm można zastosować do problemów z modelowaniem predykcyjnym klasyfikacji lub regresji[4].

Reprezentacją algorytmu jest drzewo binarne. Każdy węzeł reprezentuje pojedynczą zmienną wejściową x i punkt podziału na tej zmiennej (przy założeniu, że zmienna jest liczbowa). Liście drzewa zawierają zmienną wyjściową y, która służy do przewidywania.

Przykładowe drzewo decyzyjne zostało przedstawione na rysunku 2.8, gdzie zbiór danych to dwie dane wejściowe: wzrost w centymetrach i waga w kilogramach oraz zmienna wyjściowa – płeć (mężczyzna lub kobieta), a przykładowymi regułami są: If *Wzrost* > 180 cm Then *Mężczyzna*, If *Wzrost* <= 180 cm AND *Waga* > 80 kg Then *Mężczyzna*, If *Wzrost* <= 180 cm AND *Waga* <= 80 kg Then *Kobieta*. Przewidywanie nowych danych polega na przejściu przez drzewo od głównego węzła[4].

Uczenie się binarnego drzewa decyzyjnego jest procesem dzielenia przestrzeni wejściowej (podziały w drzewie) za pomocą algorytmu zachłannego[19] wykorzystują-

[17] ang. *Decision Trees*
[18] ang. *Classification And Regression Trees*
[19] algorytm zachłanny – algorytm, który wyznacza najlepiej rokujące w danym momencie rozwiązanie częściowe.



Rys. 2.8. Przykładowe drzewo decyzyjne.

cego dane treningowe. Wszystkie wartości są zestawiane, a różne punkty podziału są wypróbowywane i testowane za pomocą funkcji kosztu, a następnie wybierany jest podział z najniższym kosztem. Wszystkie zmienne wejściowe i wszystkie możliwe punkty podziału są oceniane i wybierane w zachłanny sposób (za każdym razem wybierany jest najlepszy punkt podziału) [4].

Dla regresji funkcja kosztu ma postać:

$$\sum_{i=1}^{n}(y_i - prediction_i)^2,$$

gdzie y jest wartością wyjściową dla próbki treningowej, a *prediction* jest to prognoza wykonana przez model dla tej próbki.

Do klasyfikacji wykorzystywana jest funkcja kosztu Gini, która wskazuje, jak czyste są węzły liści (jak mieszane są dane treningowe przypisane do każdego węzła). Funkcja kosztu Gini ma postać:

$$G = \sum_{k=1}^{n} p_k \times (1 - p_k),$$

gdzie G jest to koszt Gini we wszystkich klasach, p_k jest liczbą instancji treningowych należących do klasy k. Węzeł, który ma wszystkie klasy tego samego typu (idealna czystość klas), będzie miał G = 0, podział 50-50 klas dla problemu klasyfikacji binarnej (najgorsza czystość) będzie miał wartość G = 0,5[4].

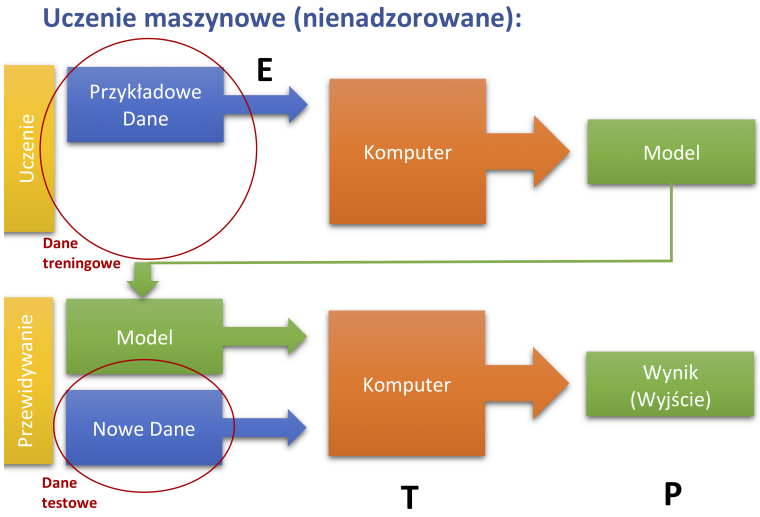
W procedurze rekurencyjnego dzielenia drzewa musi zostać podane kryterium stopu, aby algorytm wiedział, kiedy przestać dzielić, gdy przesuwa się w dół drzewa z danymi treningowymi. Najczęstszą procedurą zatrzymania jest użycie minimalnej

liczby instancji treningowych przypisanych do każdego węzła liścia. Jeśli liczba jest mniejsza niż pewne minimum, podział nie jest akceptowany, a węzeł jest traktowany jako końcowy węzeł liścia. Liczba członków szkolenia jest dostosowywana do zestawu danych, np. 5 lub 10[4].

2.2.2. Uczenie nienadzorowane[20]

Uczenie nienadzorowane zakłada dostarczanie maszynie do nauki nieoznaczonego zbioru uczącego, który składa się tylko z cech/atrybutów wejściowych (wektor $\vec{x}$) i nie zawiera etykiet. Algorytm wykorzystuje duże, zróżnicowane zbiory danych do samodoskonalenia. System uczy się modelu (funkcji h), który ma za zadanie opisać dane wejściowe (np. wyszukać wzorce w danych lub rozpoznać skupienia danych).

W uczeniu nadzorowanym system wyciąga wnioski z wcześniej podanych przykładów, natomiast w nienadzorowanym – system szuka wzorców bezpośrednio z podanego przykładu.



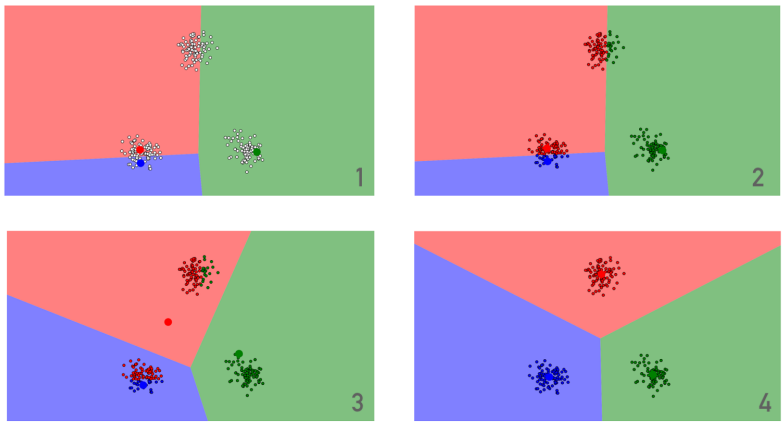
Rys. 2.9. Graficzny model przedstawiający nienadzorowany system uczący.

[20]ang. *Unsupervised Learning*

2.2.2.1. Algorytm centroidów (k-średnich[21])

Zadaniem algorytmu k-średnich jest podzielenie nieoznaczonego zbioru uczącego X = x1,. . . , xm na k klastrów (grup), tak aby każdy punkt był bardziej podobny do punktów z własnego klastra niż z punktów z innego klastra. Wyjściem algorytmu jest zestaw „etykiet" przypisujących każdy punkt danych do jednej z k grup. W grupowaniu k-średnich sposób definiowania tych grup polega na utworzeniu środka ciężkości dla każdej grupy. Centroidy są jak serce gromady, „przechwytują" najbliższe punkty i dodają je do gromady.

Algorytm polega na umieszczeniu w przestrzeni zbioru punktów ze zbioru uczącego oraz losowym umieszczeniu w przestrzeni k punktów – zwanych centroidami (Krok 1 Rys. 2.10). Następnie zostają wyliczone i przypisane do najbliższych centroidów średnie odległości między punktami (Krok 2 Rys. 2.10 — tutaj: nadanie koloru punktom zgodnie z oznaczeniem grupy). Zwykle miarą odległości między punktami jest odległość euklidesowa. Następnie zostają przesunięte centroidy na podstawie średniej arytmetycznej odległości pomiędzy wszystkimi punktami w grupie (Krok 3 Rys. 2.10 — przesunięte zostały jedynie środki, żaden punkt nie został odrzucony ani dodany do grupy). Kroki algorytmu są powtarzane aż do spełnienia określonego kryterium zbieżności (np. osiągnięcia stanu, kiedy nie ulega zmianie przynależność punktów do klas)[11]. Końcowy stan przedstawia krok 4 na Rys. 2.10.



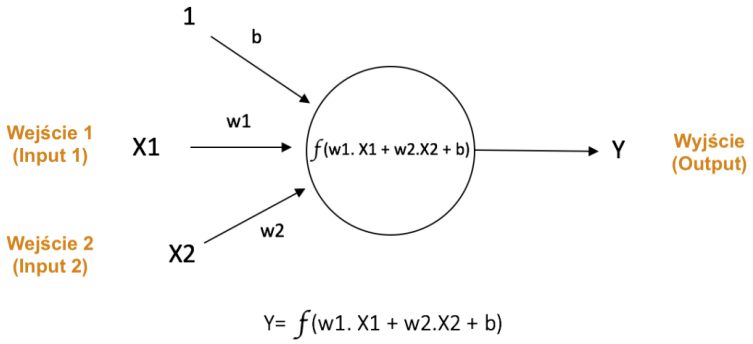
Rys. 2.10. Algorytm k-średnich przebiegu algorytmu k-średnich.
Źródło: `http://itcraftsman.pl/algorytm-k-srednich-uczenie-nienadzorowane/`

[21] ang. *k-means clustering*

2.2.2.2. Sztuczne sieci neuronowe (ANN[22])

Sztuczna sieć neuronowa to model obliczeniowy inspirowany sposobem, w jaki biologiczne sieci neuronowe przetwarzają informacje w ludzkim mózgu. Sieć neuronową można opisać jako ukierunkowany wykres, którego węzły odpowiadają neuronom, a każda (skierowana) krawędź na wykresie łączy wyjście niektórych neuronów z wejściem innego neuronu (połączenia między nimi)[4].

Podstawową jednostką obliczeniową w sieci neuronowej jest sztuczny neuron zwany perceptronem. Początkowo przyjmuje wektor $\vec{x}$ – zbiór cech wejściowych $(x_1, x_2, x_3, ...)$. Następnie neuron otrzymuje jako dane wejściowe ważoną sumę wyników neuronów połączonych jego przychodzącymi krawędziami. Neuron zwraca jedną binarną wartość wyjściową *output*. Uproszczony model perceptronu został przedstawiony na Rys. 2.11.



Rys. 2.11. Pojedynczy neuron.

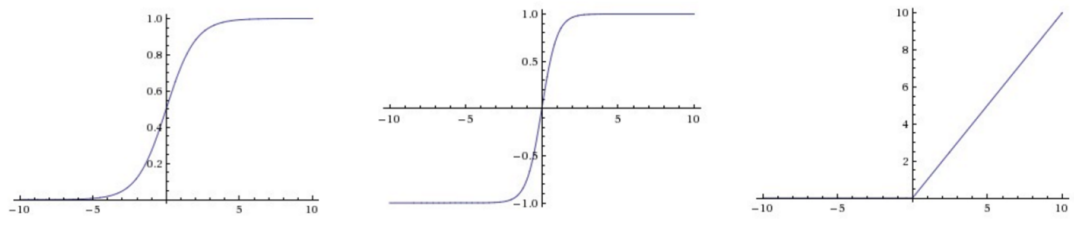
Każde wejście (x_i) ma przypisaną wagę (w_i), która jest wyznaczana na podstawie jego znaczenia w porównaniu do innych danych wejściowych. Dodatkowo istnieje wejście 1 z wagą b zwane odchyleniem - *Bias*[23]. W węźle wykonywana jest nieliniowa funkcja f (ważona suma wejść) nazywana funkcją aktywacji. Wyjście neuronu wyliczane jest ze wzoru:

$$Y = f(w_1x_1 + w_2x_2 + ... + b) = f(\sum_{i=1}^{n}, w_ix_i + b).$$

Istnieje kilka funkcji aktywacji, np.:

[22] ang. Artificial Neural Network

[23] Wprowadzenie do neuronu dodatkowego wejścia zwanego *biasem* zwiększa możliwości uczenia się - możliwe jest przesunięcie progu aktywacji w zależności od wagi odchylenia.



Rys. 2.12. Najpopularniejsze funkcje aktywacji.

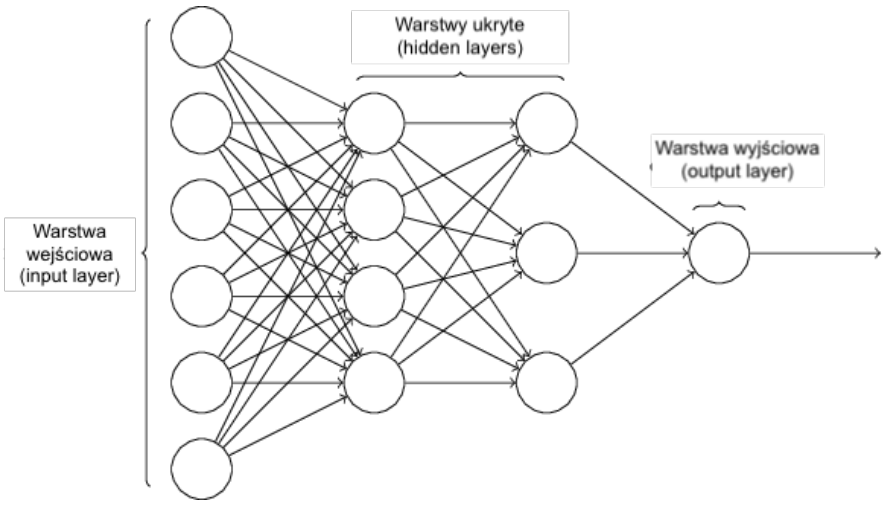
Źródło: `https://cdn-images-1.medium.com/max/1000/1*nFmfF-SAMZc9_jtFNullRg.png`

- funkcja sigmoidalna: $\sigma(x) = \frac{1}{1+e^{-x}}$ (pobiera dane wejściowe o wartościach rzeczywistych, zwraca wartości w przedziale od 0 do 1),
- tangens hiperboliczny (tanh): $tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$(pobiera dane wejściowe o wartościach rzeczywistych, zwraca wartości w przedziale od -1 do 1),
- ReLU : $f(x) = max(0, x)$ (pobiera dane wejściowe o wartościach rzeczywistych, wartości ujemna zastępuje zerem).

W sieci neuronowej, **warstwą wejściową** nazywamy skrajnie lewą warstwę. Umieszczone w niej neurony to neurony wejściowe, które dostarczają informacje ze świata zewnętrznego do ukrytych węzłów.

Warstwa wyjściowa (skrajna - po prawej) zawiera jeden lub więcej neuronów wyjściowych, które są odpowiedzialne za obliczenia i przesyłanie informacji z sieci do świata zewnętrznego.

Warstwy środkowe zwane są **warstwami ukrytymi**, ponieważ nie mają bezpośredniego połączenia ze światem zewnętrznym. Przykładowa sieć neuronowa umieszczona na Rys. 2.13 ma 4 warstwy: wejściową, wyjściową i 2 ukryte.



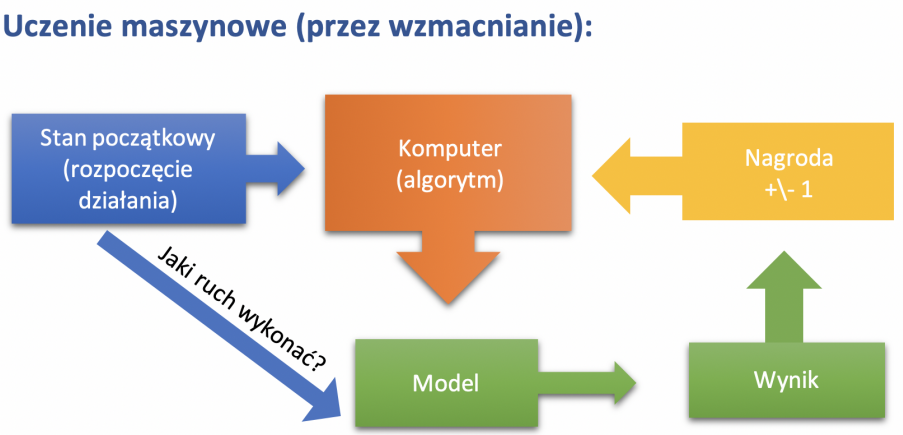
Rys. 2.13. Przykładowa wielowarstwowa sieć neuronowa.

Źródło: `https://cdn.business2community.com/wp-content/uploads/2018/02/chatbot-architecture-neural-networks.png`

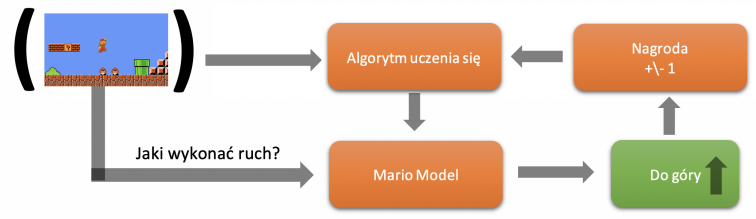
2.2.3. Uczenie przez wzmacnianie[24]

Uczenie przez wzmacnianie ma miejsce, kiedy system działa w środowisku zupełnie nieznanym. Brak jest zarówno określonych danych wejściowych jak i wyjściowych (Patrz Rys.2.14). Jedyną informacją, jaką otrzymuje maszyna ucząca się jest tzw. sygnał wzmocnienia. Sygnał ten może być albo pozytywny (nagroda) albo negatywny (kara). Metodę tę można inaczej nazwać metodą prób i błędów. Przykładem może być gra w nową grę, której reguł nie znamy. Po skończonej grze dowiadujemy się, czy wygraliśmy, czy przegraliśmy (nagroda / kara). W kolejnych turach powinien uczyć się na popełnianych błędach grach powinno iść coraz lepiej [1] (Patrz Rys.2.15). Maszyna wykorzystuje informacje zwrotne z własnych działań i doświadczeń. Celem algorytmu jest podjęcie odpowiednich działań, tak aby zmaksymalizować nagrodę w konkretnej sytuacji.

[24]ang. *Reinforcement Learning*



Rys. 2.14. Graficzny model przedstawiający system wykorzystujący uczenie przez wzmacnianie.



Rys. 2.15. Graficzny model przedstawiający system wykorzystujący uczenie ze wzmocnieniem na przykładzie gry Mario.

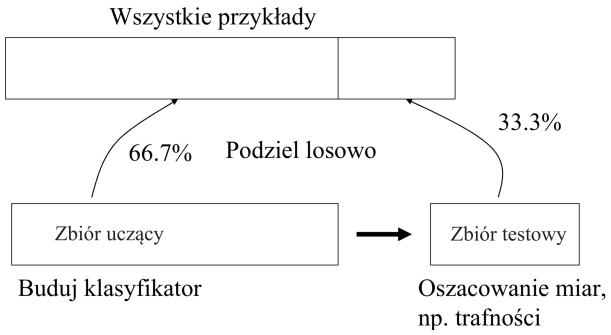
2.3. Ewaluacja modelu[25]

Ważnym krokiem w analizie predykcji jest ewaluacja modelu, czyli pomiar jego wydajności, która określa, jak dobrze model jest w stanie przewidzieć nowe (niebiorące udział w trenowaniu) dane. Poniżej zostały opisane metody oraz niektóre z metryk ewaluacji.

Metody ewaluacji:

- **Jednokrotny podział**[26] polega na "podzieleniu zbioru uczącego w proporcji 2/3 i 1/3 na kolejno zbiór uczący i zbiór testowy. Zbiór uczący służy do stworzenia modelu danych przy użyciu algorytmu uczącego, natomiast zbiór testowy służy do ewaluacji modelu."[15] (Patrz Rys. 2.16)

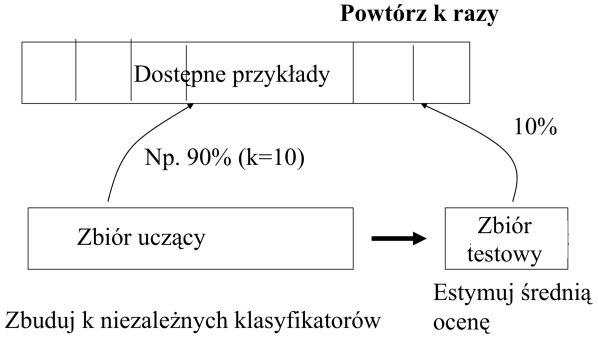
[25] ang. Model Evaluation
[26] ang. hold-out



Rys. 2.16. Jednokrotny podział – duże zbiory.

Źródło: `http://www.cs.put.poznan.pl/jstefanowski/ml/ocenaklasyfikatorow.pdf`

- **Ocena krzyżowa na k-częściach**[27] jest to odmiana metody podziału, gdzie zbiór uczący dzielimy na k części. Następnie postępujemy analogicznie jak przy jednokrotnym podziale dla każdej z k części – traktujemy k-1 części jako zbiór uczący, natomiast pozostałą część jako zbiór testowy. Cała czynność powtarzamy k razy, a końcowy wyniki jest średnią ze wszystkich ewaluacji."[15]



Rys. 2.17. Ocena krzyżowa na k-częściach – zbiory o średnich rozmiarach (od 100 do kilku tysięcy).

Źródło: `http://www.cs.put.poznan.pl/jstefanowski/ml/ocenaklasyfikatorow.pdf`

Metryki ewaluacyjne:

- **Macierz pomyłek**[28] jest metodą reprezentacji wyników predykcji w problemach klasyfikacji. Jest to tabela z czterema różnymi kombinacjami wartości przewidywanych i rzeczywistych. Czytając wierszami widzimy klasy rzeczy-

[27] ang. k-fold cross–validation
[28] ang. Confusion Matrix

wiste (prawdziwe), a kolumnami – klasy przewidywane przez model. Macierz pomyłek przedstawia nie tylko w błędy popełniane przez klasyfikator, ale co ważniejsze, pokazuje, jakie są to rodzaje błędów.

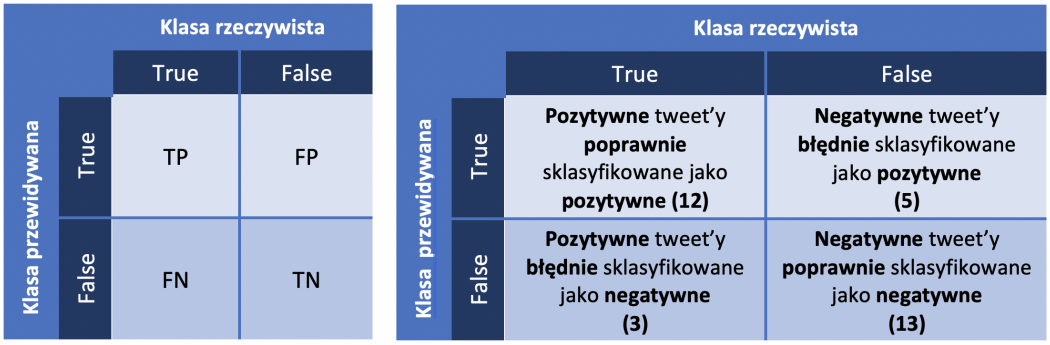
Rysunek 2.18 przedstawia ogólną postać macierzy pomyłek na przykładzie klasyfikacji tweetów, gdzie:

– *Prawdziwie pozytywy* (TP[29]): gdy rzeczywista klasa danych to P (prawda), a przewidywana jest również P (prawda).

– *Prawdziwie negatywny* (TN[30]): gdy rzeczywista klasa dancyh wynosiła N (fałsz), a przewidywana jest również N (fałsz).

– *Fałszywie pozytywny* (FP[31]): gdy rzeczywista klasa danych wynosiła N (fałsz), a przewidywana to P (prawda).

– *Fałszywie negatywy* (FN[32]): Gdy rzeczywista klasa danych wynosiła P (prawda), a przewidywana wartość to N (fałsz). [15]



Rys. 2.18. Macierz pomyłek.

Czytając tabelę wierszami mówimy o klasach rzeczywistych, kolumnami – o klasach przewidywanych przez model. Na rysunku 2.18 widać, że próbka danych zawiera 17 elementów klasy P (Pozytywne) i 16 klasy N (Negatywne). Widać też, że model przewidział poprawnie 12 elementów klasy P, natomiast 5 przypisał błędnie do klasy N, której 13 elementów przypisał poprawnie, a 3 nieprawidłowo sklasyfikował do klasy P.

[29] ang. true positive
[30] ang. true negative
[31] ang. false positive
[32] ang. false negative

- **Precyzja**[33] – "odpowiada na pytanie: „Jeżeli model przewidział, że wartość należy do danej klasy, to jakie jest prawdopodobieństwo, że ta predykcja jest poprawna?".

  Przykładowo załóżmy, że w zbiorze testowym mamy 1000 wierszy, z czego 300 należy do klasy P. Nasz model przewiduje, że do klasy P należy 450, z czego wszystkie 300 wskazał poprawnie, a 150 oszacował źle. W tym wypadku *precyzja* wynosi 0,667"[17].

$$Precision = \frac{TP}{TP + FP}$$

- **Pokrycie**[34] – "odpowiada na pytanie: „Jakie jest prawdopodobieństwo, że model przewidzi iż wartość należy do klasy T, gdy faktycznie ta wartość do niej należy".

  Dla powyższego przykładu *pokrycie* dla klasy P wynosi 100%, bo wszystkie 300 wartości ocenił poprawnie."[17].

$$Recall = \frac{TP}{TP + FN}$$

- **Dokładność/celność**[35] – "stosunek ilości poprawnie przewidzianych wartości do łącznej liczby wierszy w zbiorze testowym.

  Przykładowo mamy w zbiorze testowym 1000 wierszy, a nasz model poprawnie wskazał 650 to *dokładność* wynosi 0.65"[17].

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

[33] ang. Precision
[34] ang. Recall
[35] ang. Accuracy

2.4. Podsumowanie

W tym rozdziale zostały wyjaśnione główne pojęcia dotyczące uczenia maszynowego. Omówiono uczenie nadzorowane, nienadzorowane oraz przez wzmacnianie. Krótko opisano ich przykładowe algorytmy: maszynę wektorów nośnych, naiwny klasyfikator Bayes'a, regresja liniowa, wielomianowa i logistyczna, drzewa decyzyjne, k-najbliższych sąsiadów, k-średnich oraz sztuczne sieci neuronowe. Przedstawiono również główne metody oraz metryki ewaluacji.

Rozdział 3

Stan prac w dziedzinie

Na przestrzeni ostatnich lat zostało napisane wiele artykułów na temat analizy wydźwięku. Autorzy skupiają się na różnych etapach procesu. Na przykład Zhao Jianqiang oraz Gui Xiaolin w swoim artykule pt. *"Comparison Research on Text Pre-processing Methods on Twitter Sentiment Analysis"*[10] skupili się na porównaniu metod wstępnego przetwarzania zbioru danych. Natomiast M. Z. Asghar, A. Khan, S. Ahmad, F. M. Kundi przygotowali artykuł przeglądowy na temat technik i podejść do wydobywania cech w analizie wydźwięku w artykule pt. *"A Review of Feature Extraction in Sentiment Analysis"*[2]. W poniższym rozdziale opisano efekty badań analizy wydźwięku recenzji filmów metodami uczenia maszynowego przeprowadzone przez Bo Pang'a, Lillian Lee i Shivakumar Vaithyanathan'a w 2002r. opublikowane w artykule pt. *"Thumbs up? Sentiment Classification using Machine Learning Techniques"*[19].

3.1. Badanie analizy wydźwięku recenzji filmów

Pang, Lee i Vaithyanathan zbadali problem klasyfikacji pojedynczych zdań, nie całej dziedziny filmowej. Wykorzystali następujące algorytmy: naiwny Bayes(NB), maszyna wektorów nośnych(SVM) oraz klasyfikator maksymalnej entropii(ME).

Otrzymane przez badaczy wyniki przekroczyły linię bazową losowego wyboru wynoszącą 50% (Patrz Rys. 3.1). Każdy tekst został przekształcony na wektor cech za pomocą wektora zliczającego – metoda *worka słów*. Podczas pierwszego eksperymen-

tu jako cechy zastosowali tylko unigramy. Najwyższy wynik dokładności otrzymano za pomocą NB – 78.7% (Patrz Linia (1) Rys. 3.1).

Następnie sprawdzili, czy uwzględnienie tylko obecności cechy, a nie jej częstotliwości poprawi wynik. Znacząco wzrosła dokładność klasyfikacji dla algorytmów (Patrz Linia (2) Rys. 3.1), szczególnie dla SVM (dla pierwszego eksperymentu wynosiła 72.8%, a dla drugiego – 82.9%). W wyniku tego odkrycia, badacze nie włączali informacji o częstotliwości do NB i SVM w żadnym z kolejnych eksperymentów.

Eksperyment 3 polegał na dołączeniu bigramów do unigramów. Badacze stwierdzili, że informacje uzyskane poprzez dodanie bigramów nie mają poważnego wpływu na wyniki (Patrz Linia (3) Rys. 3.1). W kolejnym zastosowano już tylko bigramy (Patrz Linia (4) Rys. 3.1). Porównanie linii (4) z linią (2) pokazało, że poleganie tylko na bigramach powoduje spadek dokładności nawet o 5,8 punktu procentowego.

W kolejnym eksperymencie próbowali dodawać tagi do części mowy (Patrz Linia (5) Rys. 3.1). Dokładność nieznacznie się poprawia w przypadku NB, ale spada w przypadku SVM, natomiast dla ME pozostała bez zmian. Przyjrzeli się również skuteczności używania samych przymiotników. Otrzymali dosyć słabe wyniki (Patrz Linia (6) Rys. 3.1). Kolejny przeprowadzony przez nich eksperyment wykorzystujący 2633 najczęstszych unigramów jest lepszym wyborem (Patrz Linia (7) Rys. 3.1). Otrzymali dokładność klasyfikatora porównywalną do użycia wszystkich 16165 cech (linia (2)).

	Features	# of features	frequency or presence?	NB	ME	SVM
(1)	unigrams	16165	freq.	**78.7**	N/A	72.8
(2)	unigrams	”	pres.	81.0	80.4	**82.9**
(3)	unigrams+bigrams	32330	pres.	80.6	80.8	**82.7**
(4)	bigrams	16165	pres.	77.3	**77.4**	77.1
(5)	unigrams+POS	16695	pres.	81.5	80.4	**81.9**
(6)	adjectives	2633	pres.	77.0	**77.7**	75.1
(7)	top 2633 unigrams	2633	pres.	80.3	81.0	**81.4**
(8)	unigrams+position	22430	pres.	81.0	80.1	**81.6**

Źródło: `https://www.aclweb.org/anthology/W02-1011`

Rys. 3.1. Wyniki dokładności analizy wydźwięku recenzji filmów.

Ostatni przeprowadzony przez badaczy eksperyment polegał na oznaczeniu pozycji słowa w tekście zgodnie z tym, czy pojawiło się ono w pierwszej ćwiartce, ostatniej

ćwiartce, czy w środkowej połowie dokumentu. Wyniki nie różniły się zbytnio od eksperymentów z używaciem samych unigramów (Patrz Linia (8) Rys. 3.1).

3.2. Podsumowanie

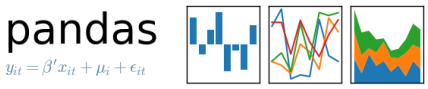
Najwyższy wynik dokładności zbioru walidującego jaki badacze osiągnęli podczas całego badania wynosił 82,9%. Do jego osiągnięcia wykorzystali algorytm SVM oraz model unigramowy. Algorytm NB zwracał słabsze wyniki w porównaniu do SVM, chociaż różnice nie były duże. Wyniki opisane w tyn rozdziale zostaną porównane z wynikami uzyskanymi podczas tworzenia projektu magisterskiego (Patrz Rozdział 5).

Rozdział 4

Narzędzia wykorzystane w projekcie

Niniejszy rozdział poświęcony został omówieniu głównych narzędzi wykorzystanych podczas realizacji projektu magisterskiego opisanego w rozdziale 5.

4.1. Pandas



Rys. 4.1. Pakiet *Pandas* – logo.
Źródło: `https://pandas.pydata.org/_static/pandas_logo.png`

Pandas to otwartoźródłowa biblioteka, zapewniająca wydajne i łatwe w użyciu struktury danych oraz narzędzia do analizy danych dla języka programowania Python.

Pakiet *Pandas* pozwala wczytywać dane z wielu źródeł, m.in. z plików tekstowych w formacie *.csv*, gdzie każdy wiersz reprezentuje wiersz arkusza kalkulacyjnego, a każda komórka wiersza arkusza kalkulacyjnego jest zwykle rozdzielona przecinkiem, średnikiem, znak tabulacji lub innym. Przykładowe użycie funkcji *read_csv*()[1],

[1] *read_csv()* odpowiada za wczytywanie plików *.csv*. Ogólną nazwę funkcji można zapisać jako: *read_«nazwa rodzaju źródła»*, np: read_csv, read_excel, read_sql, read_html, itd.

odpowiedzialnej za wczytanie danych, zostało przedstawione na rysunku 4.2. Wynikiem funkcji jest obiekt *DataFrame* (ramka danych) – dwuwymiarowa tabelaryczna struktura danych pakietu *Pandas* z oznaczonymi kolumnami i wierszami [12].

```
import pandas as pd

train_filename = './train.csv'
dataFrame = pd.read_csv(train_filename, sep='\t', header=None,
                        names=['sentiment', 'text'], encoding="ISO-8859-1"
```

Rys. 4.2. Przykładowe wczytanie pliku *.csv* za pomocą biblioteki *pandas*.

Obiekty pakietu *Pandas* dostarczają zestaw praktycznych metod matematycznych i statystycznych, np. *.sum*() oraz szereg funkcji umożliwiających operacje na danych, np. dodawanie, usuwanie albo przetwarzanie wierszy lub kolumn, sortowanie bądź grupowanie. Rysunek 4.3 przedstawia użycie funkcji *.groupby*() na obiekcie *DataFrame*. Funkcja ta dzieli duże ilości danych na zestawy w oparciu o pewne kryteria. W przykładzie etykieta *sentiment* (df['*sentiment*']) wskazuje kolumnę, która ma zostać użyta do grupowania. Aby dowiedzieć się, jak duża jest każda grupa (np. ile zawiera obserwacji), możemy użyć funkcji *.size*(), aby policzyć liczbę wierszy w każdej grupie (Patrz Rys. 4.3).

```
sentiment_df = dataFrame.groupby('sentiment').size()
print('Liczba wierszy: %d' % df.shape[0])
print(sentiment_df.head())

Out:
Liczba wierszy: 1549996
sentiment
0    775039
1    774957
```

Rys. 4.3. Przykładowe zastosowanie funkcji struktury danych *DataFrame*.

4.2. Scikit-learn

Scikit – learn (dawniej scikits.learn) to pakiet języka Python przeznaczony do uczenia maszynowego. Zawiera różne algorytmy klasyfikacji, regresji i grupowania.

Rys. 4.4. Pakiet *Scikit-learn* – logo.
Źródło: `https://scikit-learn.org/stable/_static/scikit-learn-logo-small.png`

Pakiet $Scikit-learn$ dostarcza dokumentację na temat funkcjonalności dotyczących [22]:

- klasyfikacji (określenie, do której kategorii należy obiekt),
- regresji (przewidywanie atrybutu o wartości ciągłej),
- grupowania (automatyczne grupowanie podobnych obiektów w zestawy),
- redukcji wymiarów (zmniejszenie liczby zmiennych losowych do rozważenia),
- wybór modelu (porównywanie, sprawdzanie poprawności oraz wybieranie parametrów i modeli),
- przetwarzania wstępnego (ekstrakcja cech i normalizacja).

Rysunek 4.5 przedstawia implementację modelu predykcyjnego, w którym został wykorzystany moduł $BernoulliNB$ z pakietu $scikit-learn$. W pierwszej linii zaimportowano metodę $BernoulliNB$, następnie utworzony został obiekt *bnb* typu $BernoulliNB$. Metoda $.fit()$ umożliwiła wytrenowanie modelu na danych testowych (X_train, y_train), natomiast metoda $.predict()$ – przewidzenie wyniku dla danych testowych.

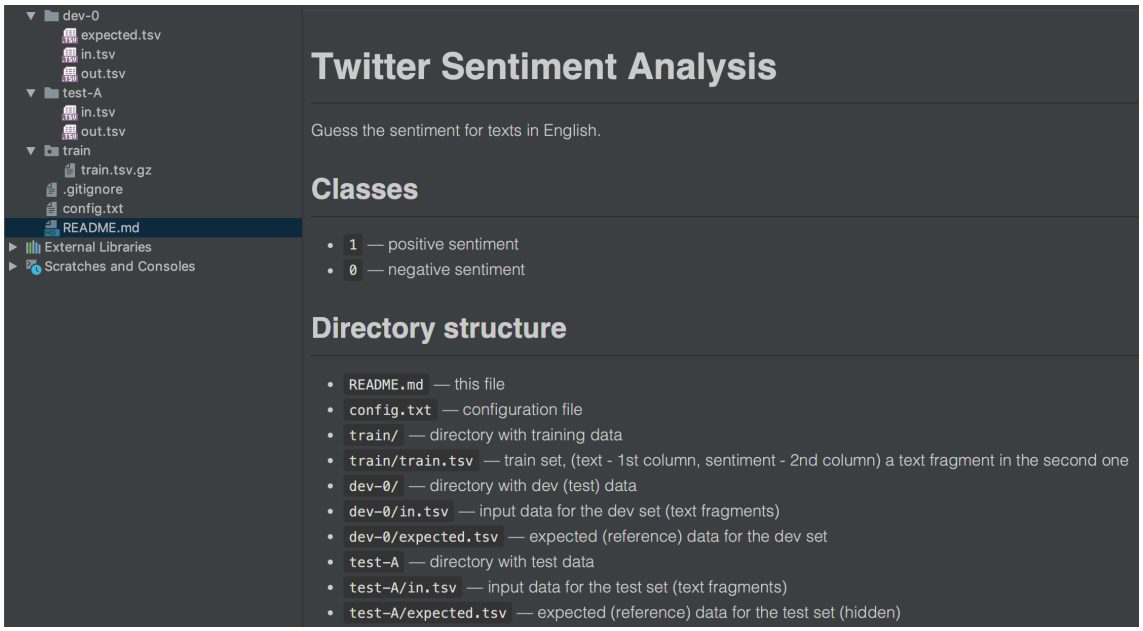
```
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB().fit(X_train, y_train)
y_pred = bnb.predict(X_test)
```

Rys. 4.5. Przykładowe zaimplementowanie modelu predykcyjnego z wykorzystaniem naiwnego algorytmu Bayesa.

4.3. GEval

GEval to narzędzie stworzone w języku Haskell umożliwiające ocenę wyników rozwiązań wyzwań związanych z uczeniem maszynowym. Dane muszą być podane w określonych plikach *.tsv* (wartości rozdzielane tabulatorami) oraz ułożone w ściśle określonej strukturze katalogów (Patrz Rys. 4.6).



Rys. 4.6. Narzędzie *Geval* – struktura katalogów wraz z opisem.

Plik *config.txt* określa, jakie metryki zostały wykorzystane podczas oceny wyników. Rysunek 4.7 przedstawia przykładowy plik konfiguracyjny, gdzie uwzględniono dwie metryki – *Likelihood* oraz *Accuracy* (dokładność).

Rys. 4.7. Plik *config.txt* narzędzia *GEval*.

Rysunek 4.8 przedstawia przykładowe uruchomienie narzędzia. *GEval* został uruchomiony na zestawie danych *dev* – 0. Dokonał oceny za pomocą metryk zadeklarowanych w wyżej wspomnianym pliku *config.txt*.

```
MacBook-Pro:twitter_sentiment_analysis user$ geval -t dev-0

Likelihood      0.54626
Accuracy        0.76857
```

Rys. 4.8. Przykład uruchomienia oceny wyników za pomocą narzędzia *GEval*.

Narzędzie *GEval* umożliwia dostęp m.in. do wyników "linia po linii", a także cech, które znacząco pogarszają wynik. Pełna dokumentacja narzędzia znajduje się pod adresem `https://gonito.net/gitlist/geval.git/`.

4.4. Gonito

Rys. 4.9. Platforma *Gonito* – logo.
Źródło: `https://gonito.net`

Gonito.net to darmowa, otwartoźródłowa platforma wspierająca naukę uczenia maszynowego. Pozwala brać udział w konkursach (wyzwaniach) mających na celu rozwiązywanie problemów związanych z danymi za pomocą metod uczenia maszynowego.

Platforma oparta jest na systemie kontroli wersji *git*[2]. Wyzwania oraz rozwiązania są przesyłane wyłącznie za jego pomocą. Platforma wykorzystuje wyżej opisane narzędzie do oceny *GEval*.

4.5. Podsumowanie

Wyżej opisane narzędzia zostały wykorzystane podczas tworzenia projektu magisterskiego. Pakiet *Pandas* umożliwił szybkie i proste wczytanie danych do struktury *DataFrame*. To na niej wykonywano podstawowe operacje m.in. umożliwiające oczyszczenie danych, czy ich podział na zbiory danych trenujących i testowych. Pakiet *scikit – learn* umożliwił otrzymanie macierzy funkcji $Tf - idf$ dla zdań. Zostały wykorzystane gotowe modele algorytmów takich jak Naiwny Klasyfikator Bayesa, Regresja Logistyczna, SVC i wiele innych, które jako parametr przyjmowały wcześniej wspomnianą macierz $Tf - idf$. Narzędzie *GEval* zostało wykorzystane do oceny wyników przewidzianych m.in. za pomocą wyuczonych modeli z pakietu *scikit – learn*. Udokumentowanie wyników ocen umożliwiła platforma *Gonito*. W czytelny sposób umożliwiła ich przegląd oraz porównywanie.

[2]System kontroli wersji – oprogramowanie służące do śledzenia zmian głównie w kodzie źródłowym. Pomaga programistom w łączeniu zmian dokonanych w plikach przez wiele osób w różnym czasie.

Rozdział 5

Projekt magisterski

5.1. Cel projektu

Celem projektu magisterskiego było porównanie wyników analizy wydźwięku *tweet'ów*[1] w języku angielskim za pomocą wybranych metod uczenia maszynowego oraz stworzenie systemu *„pyAnalysis"*, który dzięki interfejsowi użytkownika umożliwi w prosty sposób wprowadzenie i sklasyfikowanie tekstu.

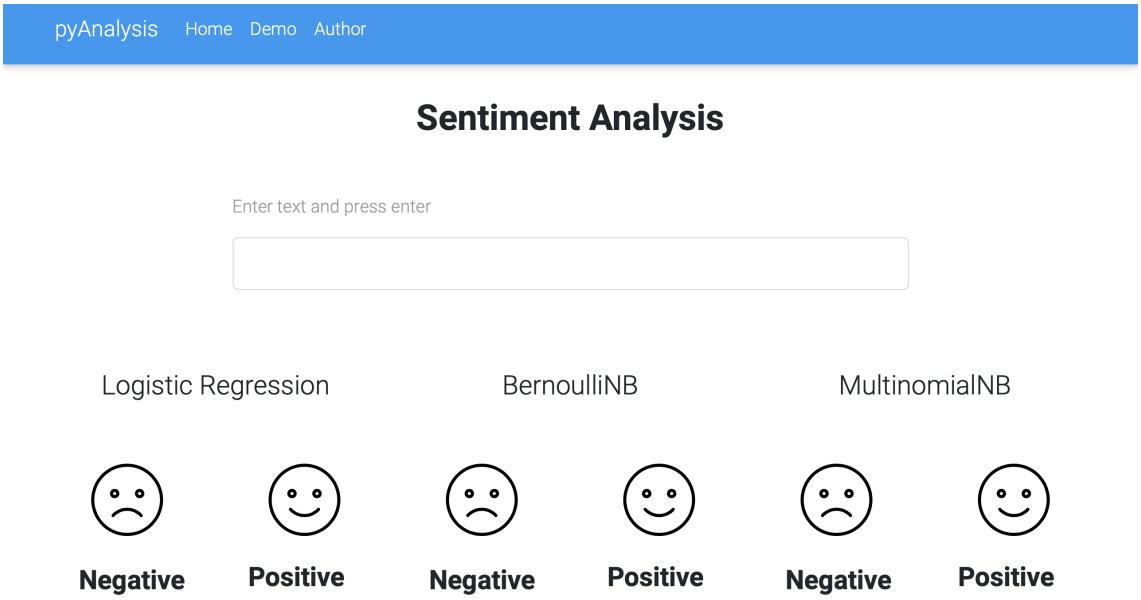
5.2. Opis systemu

W systemie *„pyAnalysis"* można wyróżnić dwie warstwy: *frontend* („warstwa odpowiedzialna za logikę obsługującą warstwę prezentacji"[25]) oraz *backend* („warstwa odpowiedzialna za logikę obsługującą warstwę logiki aplikacyjnej"[25]).

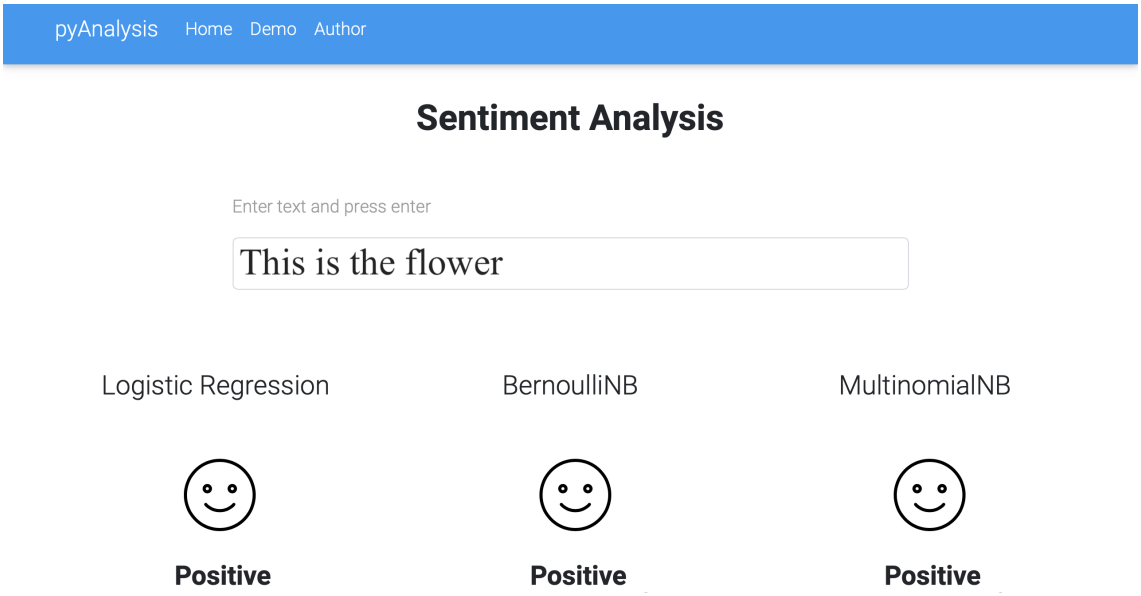
Użytkownik ma możliwość w prosty sposób przeprowadzić analizę wydźwięku dowolnego tekstu za pomocą przeglądarki stron WWW. Dane wejściowe są pobierane z prostego pola tekstowego znajdującego się na głównej stronie aplikacji (Patrz Rys. 5.1). Przekazywane są one na serwer, gdzie następuje proces klasyfikacji tekstu. Interfejs prezentuje predykcje (odpowiedź na żądanie HTTP) w postaci uśmiechniętej lub smutnej emotikony odpowiadającej pozytywnemu lub negatywnemu wydźwiękowi (Patrz Rys. 5.2). Klasyfikacja nowych tekstów w czasie rzeczywistym

[1] ang. tweet – krótka wiadomość tekstowa (maks. 280 znaków) wyświetlana na profilu zarejestrowanego autora wpisu oraz pokazywana użytkownikom, którzy obserwują dany profil w popularnym serwisie społecznościowym `https://twitter.com`.[16]

możliwa jest dzięki przeprowadzonym w ramach projektu eksperymentom opisanym w następnym podrozdziale (Patrz Rozdział 5.3.6).



Rys. 5.1. Aplikacja *„pyAnalysis”* – pole tekstowe.



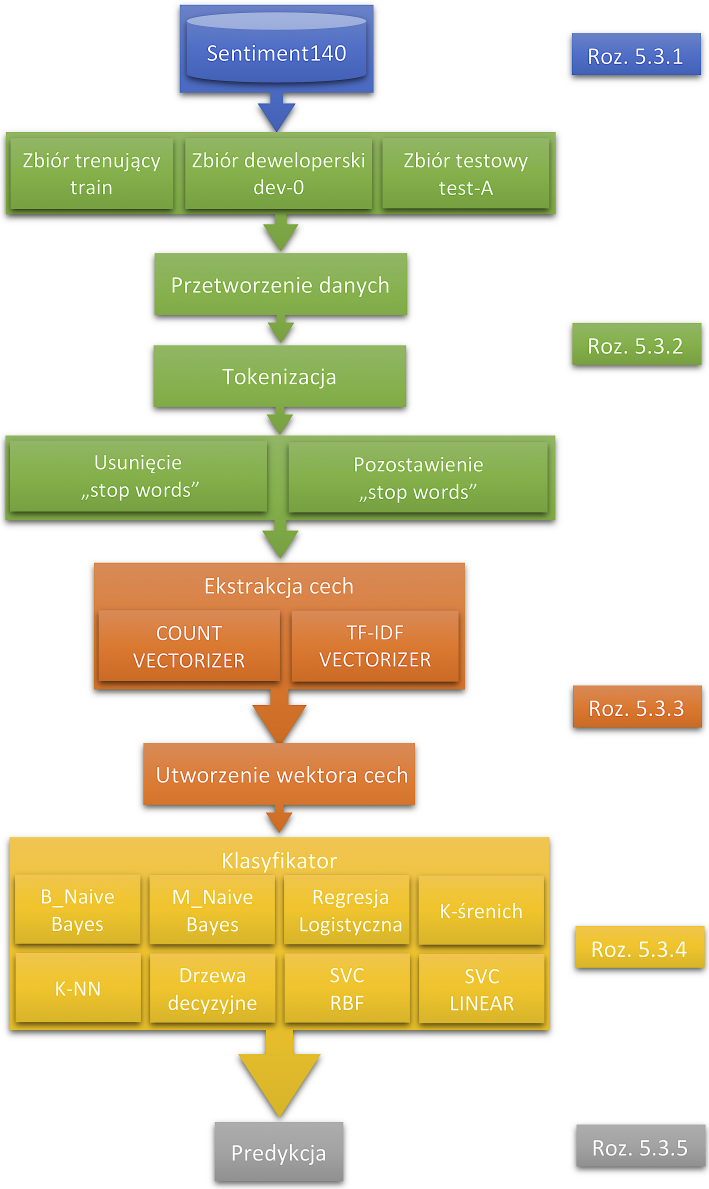
Rys. 5.2. Aplikacja *„pyAnalysis”* – predykcje.

Poniżej wymieniono technologie wykorzystane w systemie *„pyAnalysis”*:

- *frontend*: Angular, AngularCli, Typescript, HTML, CSS, Bootstrap,
- *backend*: Virtualenv, Python, Django, Django REST framework.

5.3. Proces analizy wydźwięku

Analiza wydźwięku zaliczana jest do problemów klasyfikacji, której głównym celem jest poprawne rozpoznanie nieopisanego tekstu na podstawie opisanych, znanych sobie wzorców. Możliwe jest to dzięki utworzeniu odpowiedniego procesu składającego się z przygotowania zbioru danych, utworzenia wektora cech reprezentującego w numeryczny sposób tekst, wytrenowania za pomocą zbioru uczącego klasyfikatora, predykcji zbioru walidującego i testowego oraz wyliczenia jakości naszego rozwiązania. Rysunek 5.3 przedstawia ogólny schemat takiego procesu.



Rys. 5.3. Kroki przeprowadzone podczas eksperymentów wykonanych w ramach projektu magisterskiego.

Przeprowadzone w projekcie magisterskim eksperymenty polegały na wielokrotnym wprowadzeniu co najmniej jednej zmiany w dowolnych krokach procesu, np. zmianie algorytmu uczenia maszynowego oraz sposobu ekstrakcji cech. Ich celem było porównanie skuteczności rozwiązań oraz wybranie najlepszego.

5.3.1. Źródło danych

Źródłem danych był pochodzący z Uniwersytetu Stanford zbiór *Sentiment140*. Dostępny jest on do pobrania pod adresem `http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip`. Zbiór danych składał się z 1 600 000 *tweet'ów* wyodrębnionych w automatyczny sposób za pomocą API *tweeter'a*, gdzie było po 800 000 pozytywnych i negatywnych tekstów (50:50).

Zbiór zawierał 6 kolumn:

- *target* – cel, wydźwięk *tweet'a* (0 – negatywny, 2 – neutralny, 4 – dodatni),
- *ids* – id *tweet'a* (identyfikator, np. 1467810369),
- *date* – data utworzenia *tweet'a* (np. Mon Apr 06 22:19:45 PDT 2009),
- *flag* – zapytanie (gdy nie ma – NO_QUERY),
- *user* – użytkownik, który napisał *tweet'a* (np. robotickilldozr),
- *text* – tekst *tweet'a* (np. ang. „I need a hug" – „Potrzebuję przytulenia").

Więcej informacji na temat zbioru danych można znaleźć w linku `http://help.sentiment140.com/for-students/`. Rysunek 5.4 przedstawia pierwsze 5 pozytywnych i ostatnie 5 negatywnych tekstów.

5.3.2. Przygotowanie danych

Pierwszym krokiem było podzielenie zbioru danych na zbiór uczący[2] (część stosowana w uczeniu klasyfikatora), zbiór deweloperski[3] (część stosowana do dostrojenia

[2]ang. train(ing) set
[3]ang. dev(elopment) set

	target	ids	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....
1599995	4	2193601966	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	AmandaMarie1028	Just woke up. Having no school is the best fee...
1599996	4	2193601969	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	TheWDBoards	TheWDB.com - Very cool to hear old Walt interv...
1599997	4	2193601991	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	bpbabe	Are you ready for your MoJo Makeover? Ask me f...
1599998	4	2193602064	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	tinydiamondz	Happy 38th Birthday to my boo of alll time!!! ...
1599999	4	2193602129	Tue Jun 16 08:40:50 PDT 2009	NO_QUERY	RyanTrevMorris	happy #charitytuesday @theNSPCC @SparksCharity...

Rys. 5.4. Przykładowy fragment zbioru danych *Sentiment 140*.

parametrów klasyfikatora i zapewnienia obiektywnej oceny modelu) i zbiór testowy[4] (część stosowana w późniejszej klasyfikacji na już wyuczonym klasyfikatorze). Ten zabieg był konieczny do prawidłowego korzystania z wcześniej opisanego narzędzia *GEval*.

W projekcie dokonano losowego podziału zbioru za pomocą metody *train_test_split* zaimplementowanej w module *model_selection* pakietu *sklearn* (Patrz Rys. 5.5). Parametr *test_size* odpowiada za proporcję z jaką został podzielony zbiór. Otrzymano zbiory *'train'*, *'dev-0'* i *'test-A'* o rozmiarach odpowiednio 1 549 996, 26 004 i 24 000.

```
from sklearn.model_selection import train_test_split

# podział na zbiór uczący 'train' i testowy 'test_A'
X_train, x_test_A, Y_train, y_test_A = train_test_split(text_features, text_target,
                                                        test_size=0.015, random_state=0)

# podział wcześniej wyodrębnionego'train' na nowy zb.uczący'train' i zb.deweloperski'dev_0'
X_train, x_dev_0, Y_train, y_dev_0 = train_test_split(X_train, Y_train,
                                                      test_size=0.0165, random_state=0)

train = pd.concat([Y_train.reset_index().drop(columns='index'),
                   X_train.reset_index().drop(columns='index')], axis=1)

# zbiór testowy 'test_A'
pd.DataFrame(x_test_A).to_csv('test-A/in.tsv', sep='\t', header=None)
pd.DataFrame(y_test_A).to_csv('test-A/expected.tsv', sep='\t', header=None)
# zbiór deweloperski 'dev_0'
pd.DataFrame(x_dev_0).to_csv('dev-0/in.tsv', sep='\t', header=None)
pd.DataFrame(y_dev_0).to_csv('dev-0/expected.tsv', sep='\t', header=None)
# zbiór uczący 'train'
pd.DataFrame(train).to_csv('train/train.tsv', sep='\t', header=None)
```

Rys. 5.5. Fragment kodu źródłowego opowiadający za podział zbioru danych na 'train', 'dev-0' i 'test-A'.

Kolejny krok przygotowania danych w projekcie *„pyAnalysis"* polegał na zmia-

[4] ang. test set

nie oznakowania klasy *pozytywne* z 4 na 1. Było to konieczne, aby wykorzystać zaimplementowane metryki w narzędziu *GEval*. Następnie zdekodowano fragmenty tekstu w języku HTML (np. *&amp*, *&quot*). Popularne skróty zastąpiono pełną wersją (np. *isn't* na *is not*). Usunięcie oznaczenia odpowiadania innym użytkownikom (*@nazwa_użytkownika* – odpowiedź). Mimo że zawierały pewne informacje – nie były one przydatne podczas trenowania modelu. Linki URL również zostały pominięte z tego samego powodu. Usunięto symbol '#' z hashtagów (np. ang. $\#fail$) poprzez pozbycie się wszystkich znaków innych niż litery, w tym liczb. Fragment kodu odpowiedzialny za przetworzenie danych pokazano na rysunku 5.6. Natomiast rysunek 5.7 prezentuje przykładowy fragment zbioru trenującego *'train'* przed i po oczyszczeniu.

```
from nltk.tokenize import WordPunctTokenizer

def tweet_cleaner(text):
    tok = WordPunctTokenizer()

    pat1 = r'@[A-Za-z0-9_]+'    # wzorzec oznaczenia odpowiadania innym użytkownikom
    pat2 = r'https?://[^ ]+'    # wzorzec linku URL
    www_pat = r'www.[^ ]+'      # wzorzec WWW
    combined_pat = r'|'.join((pat1, pat2))

    # Słownik negatywnych skrótów i ich pełnych wersji
    negations_dic = {"isn't": "is not", "aren't": "are not", "wasn't": "was not",
                     "weren't": "were not","haven't": "have not", "hasn't": "has not",
                     "hadn't": "had not", "won't": "will not","wouldn't": "would not",
                     "don't": "do not", "doesn't": "does not", "didn't": "did not",
                     "can't": "can not", "couldn't": "could not", "shouldn't": "should not",
                     "mightn't": "might not", "mustn't": "must not"}
    neg_pattern = re.compile(r'\b(' + '|'.join(negations_dic.keys()) + r')\b')

    # usunięcie wg wzorców
    stripped = re.sub(combined_pat, '', text)
    stripped = re.sub(www_pat, '', stripped)

    # zamiana na małe litery
    lower_case = stripped.lower()

    # zamiana negatywnych skrótów na pełne wersje
    neg_handled = neg_pattern.sub(lambda x: negations_dic[x.group()], lower_case)

    # pozbycie się wszystkich znaków innych niż litery
    letters_only = re.sub("[^a-zA-Z]", " ", neg_handled)

    # usunięcie białych znaków
    # wyodrębnienie pojedyńczych tokenów (wyrazów)
    words = [x for x in tok.tokenize(letters_only) if len(x) > 1]
    return (" ".join(words)).strip()
```

Rys. 5.6. Fragment kodu źródłowego opowiadający za przetworzenie pojedynczego *tweet'a*.

Rysunek 5.8a przedstawia 15 najczęściej występujących słów w zbiorze testowym. Najczęstszymi okazały się: „to", „the", „my", „it", ... – słowa nie wnoszące

```
                                @nileyxlove thank you
@Sweetnesss357 Thanks for the FF  Have a good ...
   @LoBosworth Post a picture with your Tweet Lo
@twitapps is there an issue with your ta_repli...
@inebriation its never a problem Mikey  I'm he...
@paulslaybaugh man, I was just thinkin about y...
```

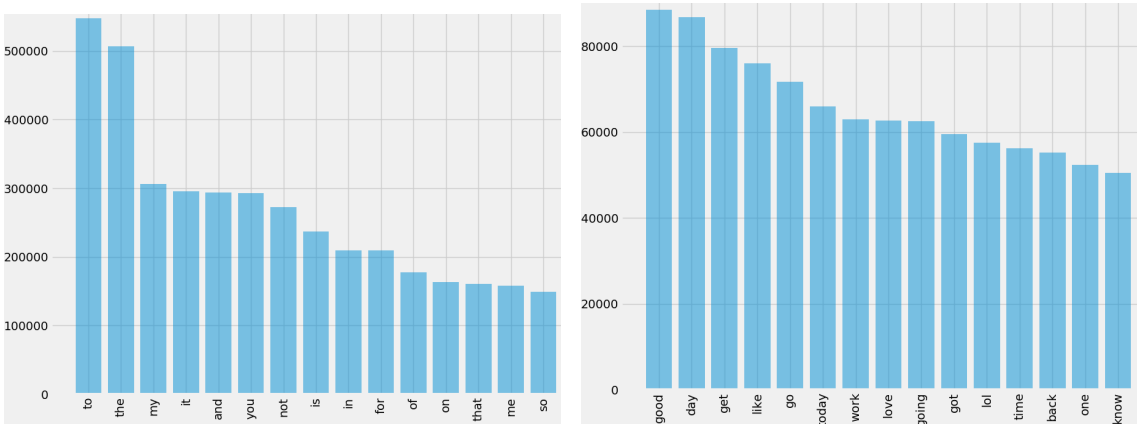
(a) Przed przetworzeniem

```
                                thank you
thanks for the ff have good weekend
    post picture with your tweet lo
is there an issue with your ta replies service...
its never problem mikey here for anyone no mat...
man was just thinkin about you
```

(b) Po przetworzeniu

Rys. 5.7. Przykładowy fragment zbioru uczącego *train*.

istotnych dla analizy wydźwięku informacji. Wszystkie z nich należą do grupy wyrazów przystankowych[5] (*stop words*). Rysunek 5.8b ukazuje 15 najczęstszych słów ze zbioru, w którym usunięto wyrazy przystankowe.



(a) Bez usunięcia wyrazów przystankowych **(b)** Po usunięciu wyrazów przystankowych

Rys. 5.8. Top 15 tokenów spośród wszystkich tweet'ów ze zbioru uczącego.

Intuicyjnym krokiem było usunięcie wyrazów przystankowych w celu poprawienia jakości modelu. Przeprowadzony został opisany dalej Eksperyment 1, aby potwierdzić tę teorię (Patrz Rozdział 5.3.6.1).

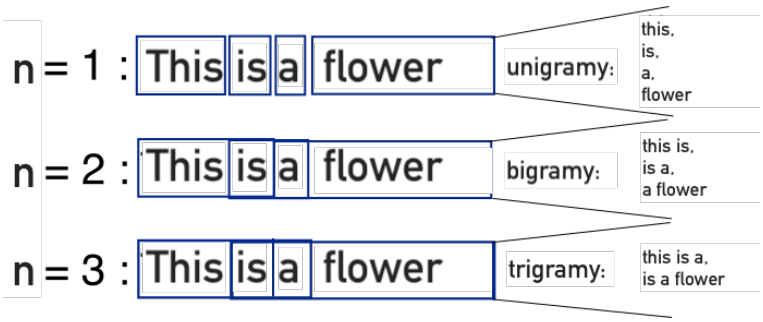
5.3.3. Ekstrakcja cech

Aby wykorzystać metody uczenia maszynowego podczas przetwarzania języka naturalnego konieczne było przekonwertowanie tekstu do postaci numerycznej. W podrozdziale 1.4.3 opisano algorytmy tworzenia wektora cech za pomocą „worka słów" oraz *Tf-idf*. Pierwszy zlicza ile razy słowo występuje w tekście (częstotliwość), natomiast drugi – opisuje trafność słów w tekście (wysoka, jeśli wyraz występuje często

[5]ang. *stop words* – zbiór nieznaczących słów typowych (takich jak „to", „czy", „i" w języku polskim, czy „be", „if", „and" w języku angielskim.)

w konkretnym tekście i bardzo rzadko gdziekolwiek indziej).

Jedną z technik wektoryzacji metodą „worka słów" jest moduł *CountVectorizer* z pakietu *scikit-learn*. Pakiet zawiera również moduł *TfidfVectorizer* wykorzystujący metodę *Tf-idf*. Oba konwertują zbiór tekstów na macierz o wymiarach: *liczba tekstów* × *liczba cech*, wykorzystując wspomniane algorytmy. Moduły te posiadają parametr *max_features*, który ogranicza liczbę cech w słowniku wybierając ograniczoną liczbę tych z najwyższą częstotliwością wystąpień. Parametr *ngram_range* pozwala wyodrębnić cechy za pomocą wybranych *n-gramów*[6], czyli ciągów występujących po sobie słów o długości *n*, poprzez określenie dolnej i górnej ich granicy *n – (min_n, max_n)*. Oba moduły za cechę domyślnie przyjmują pojedyncze słowa (unigramy, 1-gramy). W *Eksperymencie 2* (Patrz Rozdział 5.3.6.2) wykorzystano również pary (bigramy, 2-gramy) oraz trójki (trigramy, 3-gramy) dla obu modułów. Następnie porównano wyniki. N-gramy (1~3) wyodrębnione z przykładowego tekstu znajdują się na rysunku 5.9.



Rys. 5.9. N-gramy wyodrębnione z przykładowego tekstu.

Rysunek 5.10 przedstawia fragment kodu źródłowego odpowiadający za ekstrakcję cech za pomocą modułu *TfidfVectorizer*. Po zaimportowaniu modułu utworzony został obiekt *tvec* typu *TfidfVectorizer*. Przy użyciu metody *.fit()* wygenerowano słownik wraz z częstotliwością cech. Następnie przekonwertowano zbiór danych na macierz wektorów cech (funkcja *.transform()*), która została wykorzystana dalej podczas trenowania klasyfikatora.

[6] ang. n-grams

```
# zaimportowanie modułu 'TfidfVectorizer'
from sklearn.feature_extraction.text import TfidfVectorizer
import data_helper

# pobranie przetworzonych danych
x_train, y_train, x_val, y_val, x_test = data_helper.load_clean_data(remove_stopword=False)

# liczba cech
n_features = 100000
# model ngramu
ngram_range = (1, 3)

# uzyskanie obiektu modułu 'TfidfVectorizer'
tvec = TfidfVectorizer()
# ustawienie parametrów
tvec.set_params(max_features=n_features, ngram_range=ngram_range)
# utworzenie słownika wraz z częstotliwością cech
tvec.fit(x_train)
# utworzenie macierzy wektorów cech
x_train_tfidf = tvec.transform(x_train)

(...)
```

Rys. 5.10. Fragment kodu źródłowego odpowiadający za ekstrakcję cech za pomocą modułu *TfidfVectorizer* z pakietu *scikit-learn*.

5.3.4. Uczenie klasyfikatora

Algorytmy uczenia maszynowego zasilone zostały danymi treningowymi, które składały się z wektorów cech dla każdego tekstu i ich znaczników (klasa pozytywna, negatywna) w celu stworzenia modelu klasyfikacji. W projekcie do generowania modelu wykorzystano gotowe moduły z zaimplementowanymi algorytmami uczenia maszynowego znajdującymi się w pakiecie *scikit-learn*. W *Eksperymencie 3* utworzono model klasyfikacji wydźwięku za pomocą regresji logistycznej, naiwnego klasyfikatora bayesowkiego, algorytmu k najbliższych sąsiadów i innych. Następnie porównano wyniki (Patrz Rozdział 5.3.6.3).

Na rysunku 5.11 znajduje się fragment kodu odpowiadający za uczenie przykładowego klasyfikatora pochodzącego z modułu *sklearn.linear_model – LogisticRegression*. Wykorzystano do tego metodę *.fit()*, która przyjmuje jako parametr przestrzeń cech (*x_train_tfidf*) i zbiór etykiet zbioru (*y_train*) (uczenie nadzorowane). Ta metoda dostępna jest w praktycznie wszystkich algorytmach klasyfikacji w pakiecie *scikit-learn*.

```
# zaimportowanie modułu 'LogisticRegression'
from sklearn.linear_model import LogisticRegression

(...)

# przypisanie obiektu klasyfikatora do zmiennej 'clf'
clf = LogisticRegression()

# trenowanie klasyfikatora
# wykorzystując przestrzeń cech i zbiór etykiet zbioru uczącego
clf.fit(x_train_tfidf, y_train)

(...)
```

Rys. 5.11. Fragment kodu źródłowego odpowiadający za uczenie klasyfikatora z modułu *LogisticRegression*.

5.3.5. Predykcja nowych tekstów

W zależności od zestawu danych, klasyfikacja może być binarna (pozytywny lub negatywny wydźwięk) lub wieloklasowa (3 lub więcej klas). W projekcie magisterskim zastosowano klasyfikację binarną. Model po wytrenowaniu gotowy jest do klasyfikacji nowych tekstów (przypisania do klasy) za pomocą funkcji *.predict()* przyjmującej jako parametr zbiór testowy (*x_test*), który powinien mieć ten sam format, co zbiór użyty podczas wywoływania funkcji *.fit()* (Patrz Rys. 5.12).

```
(...)

# utowrzenie macierzy wektorów cech
# (ten sam kształ zbioru, co podczas trenowania modelu klasyfikatora)
x_test_tfidf = tvec.transform(x_test)

# przewidzenie wartości klas dla każdej instancji w 'x_test_tfidf'
y_test = clf.predict(x_test_tfidf)
```

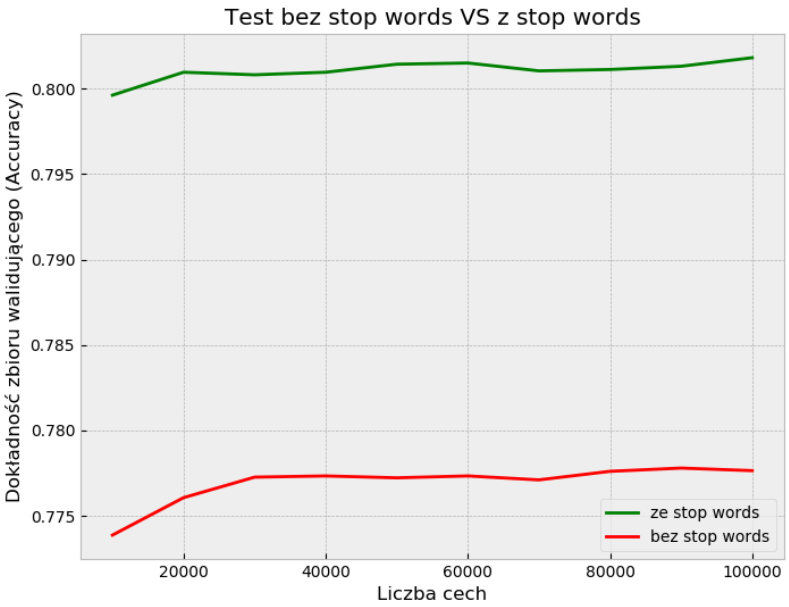
Rys. 5.12. Fragment kodu źródłowego odpowiadający za sklasyfikowanie nowych tekstów.

5.3.6. Wykonane eksperymenty

Poniżej zostaną opisane eksperymenty przeprowadzone podczas tworzenia projektu magisterskiego. Miały one na celu wyłonienie najskuteczniejszego rozwiązania problemu binarnej klasyfikacji tekstów (pozytywny lub negatywny wydźwięk). Każdy z wytrenowanych podczas badania modeli został zapisany do pliku, tak aby możliwa była klasyfikacja nowych tekstów w aplikacji *„pyAnalysis”* w szybkim czasie (bez konieczności ponownego trenowania modelu).

5.3.6.1. Eksperyment 1: Usunięcie wyrazów przystankowych

Eksperyment polegał na przeprowadzeniu testu na zbiorze uwzględniającym wyrazy przystankowe. Następnie powtórzono test na zbiorze po ich usunięciu. Wykorzystano algorytm regresji logistycznej, natomiast wektory cech utworzono przy pomocy modułu *CountVectorizer*. Test przeprowadzono na różnej liczbie cech: 10 000, 20 000, ... , 100 000. Następnie porównano wyniki w systemie *Gonito* za pomocą narzędzia *GEval*.



Rys. 5.13. Graficzne przedstawienie wyników *Eksperymentu 1*.

Rysunek 5.14 przedstawia rezultaty *Eksperymentu 1* w systemie *Gonito* uszeregowane od najwyższego wyniku dokładności zbioru walidującego. Rysunek 5.13 ukazuje te wyniki w graficzny sposób. Z wykresu wynika, że w przypadku problemu analizy wydźwięku *tweet'ów*, usunięcie wyrazów przystankowych nie poprawiło dokładności modelu. Najlepszy wynik uzyskał test wykorzystujący zbiór testowy uwzględniający wyrazy przystankowe, *CountVectorizer* z ograniczeniem cech do 100 000 oraz model regresji logistycznej (*dev-0 Accuracy:* 0.80180).

Na podstawie wyników *Eksperymentu 1*, opisane w dalszej części badania zostały przeprowadzone na zbiorach testowych uwzględniających wyrazy przystankowe.

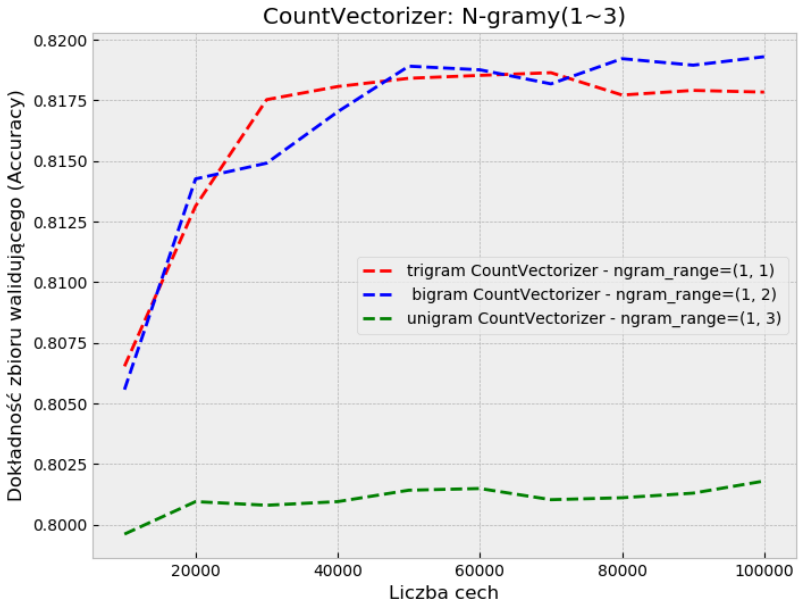
description	dev-0 Accuracy	dev-0 Likelihood	test-A Accuracy	test-A Likelihood
Wynik po 100000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80180	0.63425	0.80092	0.63221
Wynik po 60000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80149	0.63430	0.80083	0.63234
Wynik po 50000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80142	0.63415	0.80025	0.63218
Wynik po 90000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80130	0.63418	0.80092	0.63229
Wynik po 80000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80111	0.63422	0.80079	0.63227
Wynik po 70000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80103	0.63420	0.80100	0.63237
Wynik po 20000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80095	0.63421	0.80021	0.63206
Wynik po 40000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80095	0.63418	0.80017	0.63200
Wynik po 30000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.80080	0.63455	0.80029	0.63199
Wynik po 10000 cech ze stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.79961	0.63330	0.79612	0.62940
Wynik po 90000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77780	0.61757	0.77633	0.61653
Wynik po 100000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77765	0.61761	0.77654	0.61654
Wynik po 80000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77761	0.61759	0.77633	0.61655
Wynik po 40000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77734	0.61750	0.77642	0.61645
Wynik po 60000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77734	0.61770	0.77658	0.61662
Wynik po 30000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77727	0.61785	0.77696	0.61632
Wynik po 50000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77723	0.61759	0.77692	0.61657
Wynik po 70000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77711	0.61757	0.77638	0.61657
Wynik po 20000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77607	0.61731	0.77700	0.61592
Wynik po 10000 cech BEZ stopwords, CountVectorizer, ngram=(1,1), LogisticRegression	0.77388	0.61620	0.77229	0.61332

Rys. 5.14. Wyniki *Eksperymentu 1* przedstawione w systemie *Gonito*.

5.3.6.2. Eksperyment 2: Ekstrakcja cech

Na początku w eksperymencie wykorzystano moduł *CountVectorizer*. Porównano wyniki dla unigramów, bigramów oraz trigramów. Użyty został zbiór testowy uwzględniający wyrazy przystankowe oraz model regresji logistycznej. Test przeprowadzono na różnej liczbie cech: 10 000, 20 000, ... , 100 000. Następnie porównano wyniki.

Rysunek 5.15 przedstawia wykres wartości dokładności zbioru walidującego w zależności od liczby cech, natomiast rysunek 5.16 prezentuje najwyższy rezultat dokładności dla zbioru walidującego otrzymany w *Esperymencie 2*.

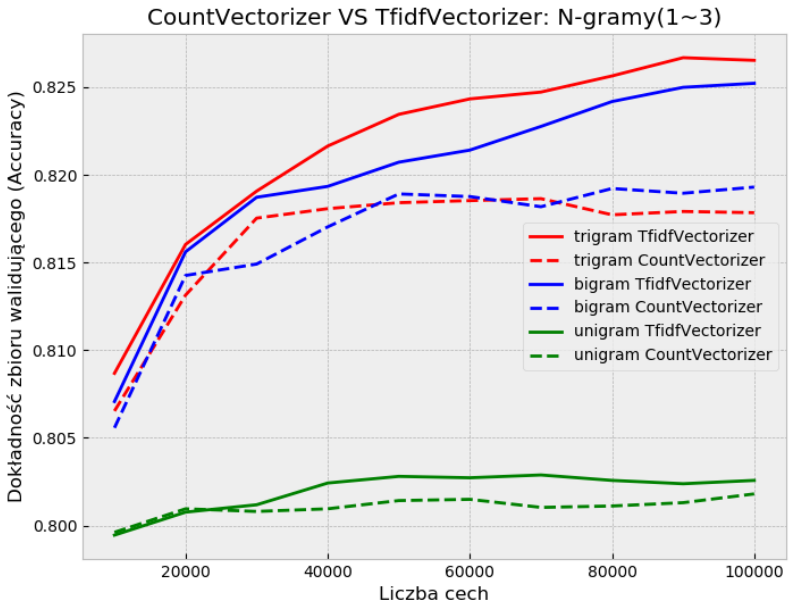


Rys. 5.15. Graficzne przedstawienie wyników *Eksperymentu 2* przy użyciu modułu *CountVectorizer*.

description	dev-0 Accuracy	dev-0 Likelihood	test-A Accuracy	test-A Likelihood
Wynik po 100000 cech ze stopwords, CountVectorizer, ngram=(1,2), LogisticRegression	0.81930	0.65335	0.82071	0.65133

Rys. 5.16. Najwyższy wynik dokładności zbioru walidującego w *Eksperymencie 2* przy użyciu modułu *CountVectorizer* w systemie *Gonito*.

Wyniki po uwzględnieniu bigramów oraz trigramów znacząco odbiegały od efektów dla samych unigramów. Eksperyment został powtórzony, ale tym razem wykorzystano moduł *TfidfVectorizer*. Następnie porównano wyniki.



Rys. 5.17. Graficzne przedstawienie wyników *Eksperymentu 2* przy użyciu modułu *CountVectorizer* i *TfidfVectorizer*.

Po przyrównaniu wartości dokładności walidacji dla różnej liczby cech i dopasowaniu danych przekształconych przez *CountVectorizer* lub *TfidfVectorizer* do regresji logistycznej zaobserwowano, że wyniki z uwzględnieniem bigramów i trigramów były znacząco wyższe niż tylko dla unigramów. W każdym przypadku n-gramu (1~3) test przeprowadzony przy użyciu modułu *TfidfVectorizer* osiągnął wyższy wyniki od modułu *CountVectorizer*. Najwyższą wartość dokładności zbioru walidującego osiągnięto wykorzystując *TfidfVectorizer*. Jako zbiór cech przyjęto (1~3)-gramy, a ich liczbę ograniczono do 90 000. Na rysunku 5.18 przedstawione zostało podsumowanie tego testu w systemie *Gonito* (dev-0 Accuracy: 0.82668).

description	dev-0 Accuracy	dev-0 Likelihood	test-A Accuracy	test-A Likelihood
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), LogisticRegression	0.82668	0.67356	0.82408	0.67078

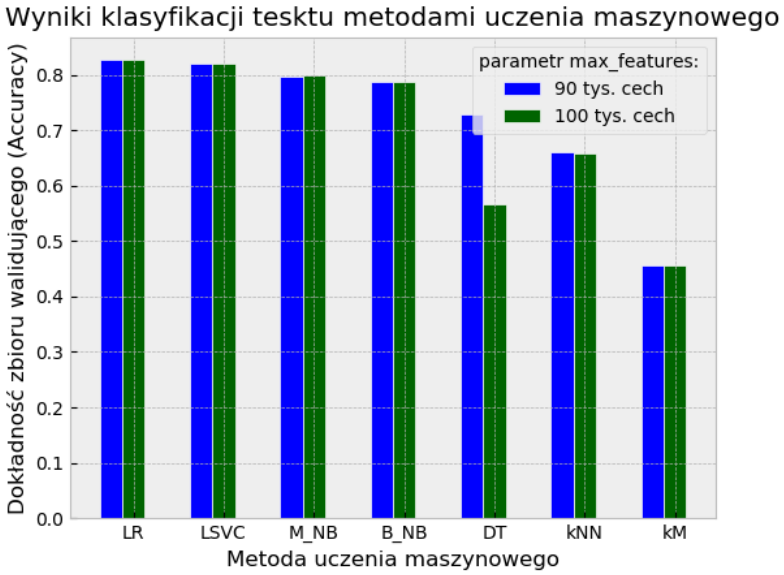
Rys. 5.18. Najwyższy wynik dokładności zbioru walidującego w *Eksperymencie 2* przy użyciu modułu *Tf-idf* w systemie *Gonito*.

Na podstawie wyników *Eksperymentu 2*, w dalszej części badań do przekształcenia danych tekstowych został wykorzystany *TfidfVectorizer* z parametrem *ngram_range* równym (1,3) – jako cechy przyjęto pojedyncze słowa, pary i trójki.

5.3.6.3. Eksperyment 3: Porównanie różnych algorytmów uczenia maszynowego

W eksperymencie użyty został zbiór testowy uwzględniający wyrazy przystankowe. Wykorzystano moduł *TfidfVectorizer*. Liczbę cech ograniczono do 90 000 i 100 000 oraz zastosowano model (1~3)-gramowy. Do klasyfikacji wykorzystano następujące algorytmy uczenia maszynowego: regresja logistyczna (LR), naiwny klasyfikator bayesowski - model wielomianowy (M_NB) i Bernoulliego (B_NB), drzewa decyzyjne (DT), liniowy klasyfikator wektorów nośnych (LSVC), algorytm k-średnich (kM), k-najbliższych sąsiadów (kNN) oraz sztuczną sieć neuronową (NN). Następnie porównano wyniki.

Z uzyskanych wartości wynikało, że klasyfikacje wykorzystujące regresję logistyczną oraz liniowy klasyfikator wektorów nośnych uzyskały najwyższe wyniki dokładności zbioru walidującego *dev-0*.



Rys. 5.19. Graficzne przedstawienie wyników klasyfikacji metodami uczenia maszynowego otrzymanych w *Eksperymencie 3*.

Zmiana liczby cech nie miała większego wpływu na wyniki. Jedynie dla drzew decyzyjnych dokładność zbioru walidującego znacząco spadła (Patrz Rys. 5.19).

Na rysunkach 5.20 i 5.21 zostały przedstawione wyniki Eksperymentu 3 uszeregowane malejąco według kolumny *„dev-0 Accuracy”*.

description	dev-0 Accuracy	dev-0 Likelihood	test-A Accuracy	test-A Likelihood
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), LogisticRegression	0.82653	0.67385	0.82467	0.67126
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Linear_SVC	0.82018	0.12615	0.82350	0.13107
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Multinomial_NB	0.79795	0.64428	0.80058	0.64544
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Bernoulli_NB	0.78761	0.47338	0.78712	0.47164
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Nearest_Neighbors3	0.65901	0.37016	0.66104	0.38064
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Nearest_Neighbors5	0.63267	0.31527	0.63433	0.32120
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Nearest_Neighbors7	0.62556	0.40794	0.62754	0.40286
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Nearest_Neighbors2	0.61348	0.26318	0.61479	0.27322
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Decision_Tree	0.56653	0.51267	0.56800	0.51311
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), KMeans	0.45658	0.00192	0.45242	0.00183

Rys. 5.20. Wyniki *Eksperymentu 3* przedstawione w systemie *Gonito* (ograniczenie liczby cech do 100 tys.).

description	dev-0 Accuracy	dev-0 Likelihood	test-A Accuracy	test-A Likelihood
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), LogisticRegression	0.82668	0.67356	0.82408	0.67078
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Linear_SVC	0.82041	0.12649	0.82212	0.12901
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Multinomial_NB	0.79703	0.64342	0.79938	0.64442
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Bernoulli_NB	0.78673	0.47508	0.78692	0.47318
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Decision_Tree	0.72950	0.04761	0.73088	0.04868
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), Nearest_Neighbors3	0.66071	0.36590	0.66221	0.37560
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), KMeans	0.45647	0.00192	0.45192	0.00182

Rys. 5.21. Wyniki *Eksperymentu 3* przedstawione w systemie *Gonito* (ograniczenie liczby cech do 90 tys.).

5.4. Wnioski z eksperymentów

Cel projektu został osiągnięty. Stworzono system *„pyAnalysis”* dzięki któremu użytkownik może sklasyfikować wydźwięk wpisanego tekstu za pomocą metod uczenia maszynowego. Przeprowadzone w ramach projektu eksperymenty pokazały, że zastosowanie zbioru uczącego uwzględniającego wyrazy przystankowe poprawia dokładność zbioru walidującego. Najwyższe wyniki uzyskano wykorzystując do wektoryzacji cech moduł *TfidfVectorizer*. Przyjmował on jako cechę unigramy, bigramy oraz trigramy.

Rysunek 5.22 przedstawia trzy testy, które uzyskały najwyższy wynik dokładności walidacji. Zaobserwowano, że klasyfikator regresji logistycznej przewyższa inne algorytmy. Najniższe wyniki osiągnięto wykorzystując metodę k-średnich.

Wyniki ewaluacji, uzyskane w ramach eksperymentów w projekcie magisterskim, odbiegają od wyników wcześniej opisanego badania Pang'a, Lee i Vaithyanathan'a (Patrz Rozdział 3). Badacze osiągnęli najwyższy wynik wykorzystując jako cechy tylko unigramy. W ich przypadku dołączenie bigramów nie miało poważnego wpływu na wyniki. W eksperymentach przeprowadzonych w ramach projektu magisterskiego uzyskano odwrotny efekt. Dodanie bigramów, a nawet trigramów znacząco poprawiło wynik dokładności walidacji.

Oba eksperymenty polegały na binarnym sklasyfikowaniu wydźwięku tekstów metodami uczenia maszynowego. W jednym i drugim Naiwny Algorytm Bayes'a zwracał słabsze wyniki niż maszyna wektorów nośnych. Znaczącą różnicą była wielkość korpusu. Badacze wykorzystali 752 negatywnych i 1301 pozytywnych recenzji filmów, natomiast w projekcie magisterskim – po 800 000 pozytywnych i negatywnych *tweet'ów*.

description	dev-0 Accuracy	dev-0 Likelihood	test-A Accuracy	test-A Likelihood
Wynik po 90000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), LogisticRegression	0.82668	0.67356	0.82408	0.67078
Wynik po 100000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), LogisticRegression	0.82653	0.67385	0.82467	0.67126
Wynik po 80000 cech ze stopwords, TfidfVectorizer, ngram=(1,3), LogisticRegression	0.82564	0.67325	0.82392	0.67029

Rys. 5.22. Top3 wyniki dokładności walidacji przedstawione w systemie *Gonito*.

Rozdział 6

Podsumowanie

Coraz częściej ludzie wyrażają swoje opinie na temat produktów, usług itp. w mediach społecznościowych. Wzrost zainteresowania opinią publiczną skłania do przeprowadzania analizy wydźwięku nastrojów za pomocą metod przetwarzania języka naturalnego. W przyszłości rozwój analizowanie wydźwięku *tweet'ów* na konkretny temat może mieć miejsce w czasie rzeczywistym. Z takiej aplikacji będą mogli korzystać na przykład inwestorzy pragnący dowiedzieć się, co ludzie uważają na temat wprowadzonego na rynek nowego produktu.

Celem projektu magisterskiego opisanego w niniejszej pracy było porównanie wyników analizy wydźwięku tekstów w języku angielskim wykorzystującej algorytmy uczenia maszynowego. Przeprowadzone eksperymenty polegały na wielokrotnym wprowadzeniu co najmniej jednej zmiany w dowolnych krokach procesu przewidywania wydźwięku, np. zmianie algorytmu uczenia maszynowego oraz sposobu ekstrakcji cech. Dla opisanego w pracy problemu klasyfikacji *tweet'ów* zaobserwowano podczas eksperymentów, że usunięcie wyrazów przystankowych nie poprawiło dokładności zbioru walidującego. Znacznie lepsze wyniki uzyskano po wprowadzeniu bigramów i trigramów do zestawu cech trenujących. Najlepszym spośród przetestowanych algorytmów wektoryzacji cech okazał się Tf-idf.

Podczas przeprowadzania eksperymentów wytrenowane modele klasyfikatorów zostały zapisane do plików. Umożliwiło to klasyfikacje nowych tekstów w systemie *"pyAnalysis"* w czasie rzeczywistym (długotrwały proces uczenia został zastąpiony wczytaniem z pliku wytrenowanego wcześniej modelu).

W przyszłości system można będzie rozwinąć o funkcjonalność analizy wydźwięku ostatnio dodanych opinii na temat konkretnej usługi, produktu, itp. z różnych portali internetowych. Użytkownicy mogliby na bieżąco sprawdzać, jaki jest stosunek innych osób do danego tematu.

Spis rysunków

Bibliografia

[1] Co to jest „machine learning”?, 2018.

[2] Asghar, M. Z., Khan, A., Ahmad, S., and Kundi, F. M. A review of feature extraction in sentiment analysis. *Journal of Basic and Applied Scientific Research* (2014).

[3] Bollen, J., Mao1, H., and Zeng, X.-J. Twitter mood predicts the stock market. *Journal of Computational Science* (2010).

[4] Brownlee, J. *Master Machine Learning Algorithms. Discover How They Work and Implement Them From Scratch.* Machine Learning Mastery, 2016.

[5] Brownlee, J. Simple linear regression tutorial for machine learning, 2016.

[6] Grubich, A. Czym jest tf-idf i jak wpływa na pozycjonowanie strony. *Sunrise System* (2019).

[7] Grus, J. *Data science od podstaw. Analiza danych w Pythonie.* Helion S.A., 2018.

[8] Horzyk, A. Metody inteligencji obliczeniowej, 2015.

[9] Issenberg, S. How obama's team used big data to rally voters. *MIT Technology Review* (2012).

[10] Jianqiang, Z., and Xiaolin, G. Comparison research on text pre-processing methods on twitter sentiment analysis. *IEEE Access* (2017).

[11] Kondas, A. Algorytm k-średnich – uczenie nienadzorowane, 2016.

[12] Ksopyla. Python pandas i wizualizacja danych. *About Data* (2016).

[13] Liu, B. Sentiment analysis and opinion mining. *Morgan & Claypool Publishers* (2012).

[14] Mitchell, T. M. *Machine Learning.* McGraw-Hill Science/Engineering/Math, 1997.

[15] Narkhede, S. Understanding confusion matrix. *Towards Data Science* (2018).

[16] Nations, D. What is a tweet on twitter? *Lifewire* (2018).

[17] Owsianik, P. Machine learning workflow cz 3 – testowanie modelu. *Thinking in code* (2017).

[18] Pagolu, Challa, Panda, and Majhi. Sentiment analysis of twitter data for predicting stock market movements. *International conference on Signal Processing, Communication, Power and Embedded System* (2016).

[19] Pang, B., Lee, L., and Vaithyanathan, S. Thumbs up? sentiment classification using machine learning techniques. *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (2002).

[20] Pazdanowski, M. Aproksymacja. Master's thesis, Politechnika Krakowska, 2012.

[21] Potoniec, J. Naiwny klasyfikator bayesa.

[22] scikit-learn developers. Scikit-learn user guide, 2019.

[23] Shalev-Shwartz, S., and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

[24] Wojnowski, J. *Wielka encyklopedia PWN.* Państwowe Wydawnictwo Naukowe PWN, 2005.

[25] Łukasz Walkiewicz. Ogólna architektura aplikacji warstwowych. *Trzecia kawa* (2014).