Uniwersytet Zielonogórski

Wydział Informatyki, Elektrotechniki i Automatyki

Praca dyplomowa

Kierunek: Informatyka

System do klasteryzacji dokumentów tekstowych

Kacper Drapała

Promotor:
Dr inż. Marek Kowal

Pracę akceptuję:

.....
(data i podpis promotora)

Zielona Góra, Luty 2018

Streszczenie

Niniejsza praca przedstawia realizację projektu, którego celem jest opracowanie systemu, który w sposób nienadzorowany będzie w stanie skategoryzować dokumenty tekstowe na podstawie ich treści.

Na początku pracy przedstawiona została teoria dotycząca eksploracji danych. Opisana została ścieżka instalacji oraz konfiguracji środowiska pracy i wykorzystanego języka programowania. Omówiono możliwości wykorzystanych bibliotek do wstępnego przetwarzania i klasteryzacji dokumentów tekstowych oraz wykorzystanych w dalszej części metod i technik do ich obróbki.

W kolejnej części znajdują się szczegółowe informacje dotyczące wybranych technologii do uzyskania zamierzonego celu. Wytłumaczone zostały poszczególne etapy przygotowania dokumentów tekstowych, m.in. czyszczenie, normalizacja oraz transformacja dokumentów tekstowych.

Ostatni etap pracy poświęcono na badania eksperymentalne, których celem było zweryfikowanie skuteczności przygotowanego systemu. Badania przeprowadzono na zbiorze rzeczywistych dokumentów tekstowych pochodzących z ponad 2000 źródeł wiadomości.

Słowa kluczowe: sztuczna inteligencja, dokumenty tekstowe, klasteryzacja

Spis treści

Spis rysunków

Spis tabel

Spis listingów

Rozdział 1

Wstęp

1.1. Wprowadzenie

Temat niniejszej pracy nie jest przypadkowy. Zastosowanie sztucznej inteligencji z dnia na dzień staje się coraz popularniejsze. Wielkie firmy wykorzystują ją do szybkiego rozpoznawania między innymi obrazów, dźwięków lub tekstów.

Inspiracją do napisania takiego systemu jest ciągła konieczność poprawiania algorytmów, szukających takich samych lub podobnych produktów ze sklepów internetowych w aktualnie wykonywanej pracy zawodowej. Problemem jest tu zbyt duża różnorodność dotycząca kategorii danych produktów. Towar z jednej kategorii ma w opisie inne informacje niż produkt z drugiej kategorii. System do klasteryzacji dokumentów tekstowych pozwoli na częściowe rozwiązanie tego problemu. Dzięki temu można w łatwy sposób wygenerować listę dokumentów tekstowych, które będą składać się z opisów, nazw, kategorii czy specyfikacji produktów. Następnie należałoby uruchomić algorytm, który pogrupuje dokumenty tekstowe w takie grupy, w których obiekty będą do siebie jak najbardziej podobne.

System ten będzie bardzo przydatny w branży E-Commerce, której pracownicy będą szybciej analizować ceny na rynku z różnych sklepów internetowych czy kategorii danych produktów. System sprawdzi się również do grupowania innych rodzajów dokumentów tekstowych. W poniższej pracy wykorzystam go do pogrupowania artykułów powiązanych ze sobą tematycznie, co przedstawię w kolejnych rozdziałach.

1.2. Cel i zakres pracy

Celem pracy jest utworzenie systemu, który na wejściu otrzymuję listę dokumentów tekstowych, a na wyjściu generuje grupy tych dokumentów dokonując selekcji poszczególnych dokumentów do grup na podstawie treści dokumentów. Oznacza to, że odległość pomiędzy dokumentem tekstowym z jednej grupy, a dokumentem z innej musi być jak największa, natomiast dla obiektów znajdujących się w tej samej grupie - jak najmniejsza.

Praca będzie swym zakresem obejmowała:

- dokonanie przeglądu bibliotek do wstępnego przetwarzania i klasteryzacji dokumentów tekstowych,
- przygotowanie testowego zbioru dokumentów tekstowych,
- czyszczenie, normalizacja oraz transformacja dokumentów tekstowych,
- zastosowanie algorytmów klasteryzacji do budowy systemu klasteryzacji treści dokumentów tekstowych,
- wykonanie testów weryfikujących skuteczność opracowanego systemu.

1.3. Struktura pracy

Praca została podzielona na dwie części, teoretyczną oraz praktyczną. Część teoretyczna została zaprezentowana w rozdziale 2. W ramach tej części opisano metody przetwarzania dokumentów tekstowych, klasteryzacji oraz sposobów weryfikacji wyników. W rozdziale 3 zaprezentowano przykładowe wyniki dla metod, które opisano w rozdziale 2. Wyniki przedstawione są w formie rysunków oraz tabel wraz z komentarzem.

Rozdział 2

Założenia projektowe

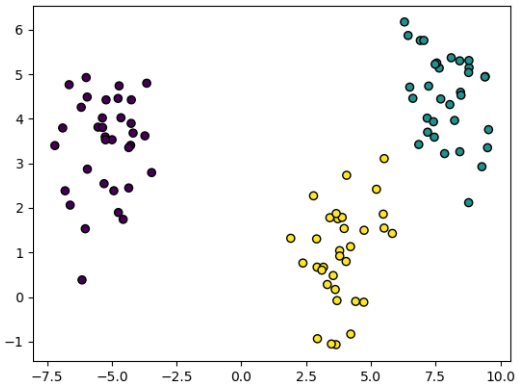
2.1. Eksploracja danych

Eksploracja danych (ang. data mining) jest dziedziną nauki zajmującą się metodami pozyskiwania wiedzy z danych. Główne metody eksploracji danych to klasteryzacja i klasyfikacja danych, odkrywania wzorców asocjacji lub sekwencji, selekcja danych odstających. Algorytmy wykorzystywane do eksploracji danych pochodzą głównie z zakresu statystyki i uczenia maszynowego [1]. Zwykle, aby wydobyć wiedzę np. z tekstu, musimy przetworzyć ogromne ilości danych. Źródłem tych danych mogą być strony WWW, artykuły w gazetach czy wiadomości E-mail. Dzięki eksploracji można wydobyć kolejno: informacje o konkretnym produkcie ze strony WWW, określić treść dokumentów tekstowych oraz określić czy dany E-mail jest spamem.

2.1.1. Klasteryzacja

Klasteryzacja (grupowanie) inaczej nazywana jest klasyfikacją bez nadzoru. To metoda, dzięki której następuje pogrupowanie pewnego zbioru elementów w taki sposób, aby dane poszczególnych grup były do siebie jak najbardziej podobne. Grupowanie dąży do tego, aby odległości pomiędzy obiektami w jednej grupie były jak najmniejsze, natomiast odległość pomiędzy grupami obiektów - jak największa.

Na wykresie Rys.2.1 osadzone są pewne obiekty. Trzema różnymi kolorami oznaczono przynależność do grupy. Możemy zaobserwować, że obiekty należące do tej samej grupy są skupione wokół siebie.



Rys. 2.1. Pogrupowane obiekty

2.2. Python

Python jest wieloplatformowym językiem programowania wysokiego poziomu. Został stworzony w celu zapewnienia dużej czytelności kodu źródłowego. Składnia tego języka jest przejrzysta dzięki zminimalizowaniu konstrukcji składniowych. Korzysta on z wcięć zamiast klamr, które oznaczają początek i koniec bloku instrukcji. Python rozwijany jest jako otwarty projekt (ang. open-source). Instalacja Pythona sprowadza się do pobrania instalatora oraz uruchomienia go, pamiętając, aby podczas instalacji zaznaczyć zgodę na dodanie przez instalator ścieżki Pythona do zmiennych środowiskowych systemu. Do realizacji niniejszej pracy wykorzystano Pythona w wersji 3.6.4.

Weryfikacja poprawności instalacji języka programowania sprowadza się do uruchomienia programu z wiersza poleceń systemu Windows. Wówczas uruchamiamy program Python, w którym już teraz jest możliwość pisania mało skomplikowanych programów.

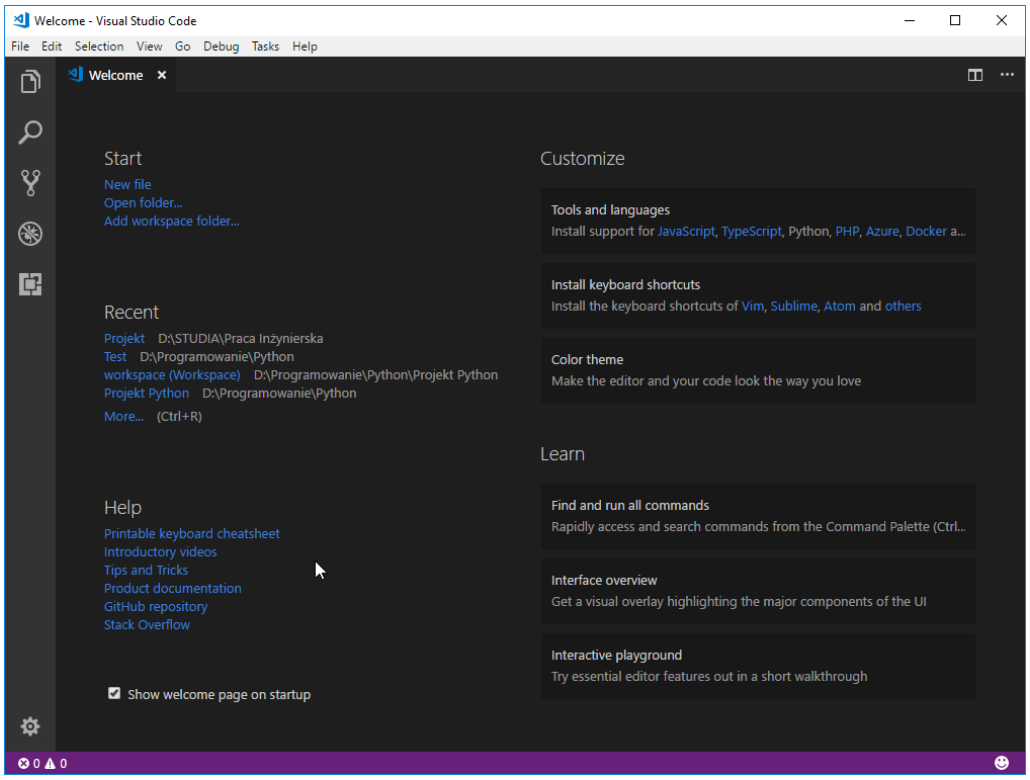
```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> c = a + a * a
>>> print(c)
6
>>>
```

Rys. 2.2. Prosty przykład programu napisanego w języku Python w wierszu poleceń

2.3. Środowisko pracy

Projekt powstał w środowisku Visual Studio Code w wersji 1.19.2 od firmy Microsoft oraz został napisany w języku Python w wersji 3.6.4. Środowisko to dostępne jest na platformach między innymi Windows, Linux i macOS. Środowisko jest darmowe oraz open-source.

Instalacja środowiska programistycznego na platformie Windows przebiega w bardzo prosty sposób. Pierwszym etapem jest instalacja Visual Studio Code, czyli edytora kodu, który wspiera najważniejsze operacje programowania. Tymi operacjami są na przykład debugowanie czy system kontroli wersji. Środowisko to zostało stworzone z myślą o szybkich cyklach pisania kodu oraz debugowania. Podczas instalacji należy zaznaczyć pole, które doda ścieżkę oprogramowania do zmiennych środowiskowych systemu operacyjnego.



Rys. 2.3. Interfejs środowiska Visual Studio Code

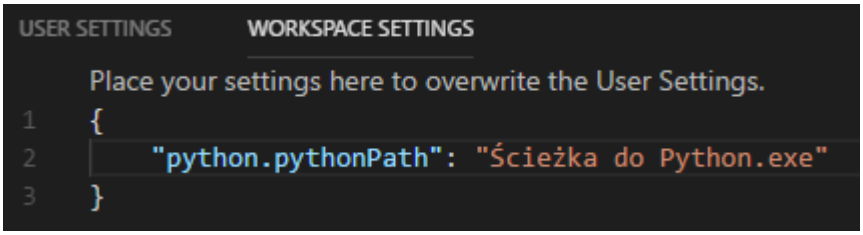
Kolejnym etapem jest konfiguracja wykorzystywanego środowiska, by móc uruchomić pisane w języku Python programy. W tym celu należy utworzyć nowy folder, który jest odpowiednikiem projektu w innych środowiskach programowania. Następnie stworzyć plik o rozszerzeniu *.py*. W tym momencie pojawi się komunikat o za-

instalowaniu rozszerzenia do Visual Studio Code, z dodatkami do języka Python, takimi jak:

- formatowanie kodu,
- refaktoryzacja,
- pilnowanie składni języka,
- podpowiedzi.

Należy go zainstalować oraz zresetować Visual Studio Code klikając przycisk "Reload" obok dodanego rozszerzenia.

Następnie wchodzimy w zakładkę "Debug (Ctrl+Shift+D)" i klikamy na rozwijaną listę, po czym wybieramy opcję "Add Configuration...". W tym miejscu na górze programu wyświetlą się propozycje zainstalowanych w systemie środowisk. Po dodaniu konfiguracji Python, utworzy nowy folder o nazwie ".vscode" oraz plik "launch.json", w którym są wszystkie konfiguracje programu. Najważniejszą konfiguracją ze wszystkich jest ta położona na samej górze z nazwą "Python", ponieważ bez niej nie będziemy w stanie uruchamiać programów. Ostatnim elementem konfiguracji jest dodanie ścieżki dostępu do języka programowania. W zakładce "Explorer (Ctrl+Shift+E)" należy kliknąć prawym przyciskiem myszy i wybrać "Open Folder Settings". Utworzy się nowy plik w folderze ".vscode" o nazwie "settings.json", w którym są wszelkie opcje projektu. Po uruchomieniu tego pliku klikamy po prawej stronie "Workspace Settings", gdzie podajemy ścieżkę do programu Python.



Rys. 2.4. Absolutna ścieżka do zainstalowanego programu Python.exe

Ostatnim krokiem jest weryfikacja, czy wszystko zostało poprawnie skonfigurowane. W tym celu wystarczy otworzyć stworzony wcześniej plik z rozszerzeniem *.py* i napisać pierwsze standardowe polecenie "print('Hello World!')".

2.4. Biblioteki do wstępnego przetwarzania dokumentów tekstowych

Aby zrealizować zaplanowane zadanie klasteryzacji dokumentów konieczne jest wstępne przetworzenie dokumentów tekstowych do postaci akceptowalnej przez procedury klasteryzacji. Język Python oferuje szereg bibliotek, które wspomagają realizację tych procesów. Najczęściej wykorzystywane do takich zadań biblioteki to Scikit-learn [2] oraz Natural Language Toolkit [3].

2.4.1. Scikit-learn

Scikit-learn jest darmową biblioteką do wykonywania czynności związanych z uczeniem maszynowym. Wprowadza wiele rozwiązań, jak: algorytmy klasyfikacji, regresji czy klasteryzacji.

Biblioteka wymaga dodatkowo kilka innych modułów by działać poprawnie. Tymi modułami są:

- NumPy w wersji wyższej bądź równej 1.8.2
- SciPy w wersji wyższej bądź równej 0.13.3

Instalacja bibliotek odbywa się z wiersza poleceń. Istnieją dwa sposoby realizacji tej procedury. Pierwszy z nich polega na uruchomieniu wiersza poleceń z samego Windowsa. W drugim instalacja odbywa się poprzez Terminal w Visual Studio Code, który ma wbudowaną w sobie integrację z Windows PowerShell. Jest to odpowiednik wiersza poleceń.

Kolejnym etapem jest instalacja wymaganych modułów dla języka Python. Instalację można przeprowadzić z wykorzystaniem menadżera pakietów "pip" wydając następujące polecenia: "pip install nazwa modułu", dla wyżej wymienionych bibliotek:

Listing 2.1. Instalacja wymaganych modułów z podaniem wersji

```
1 pip install numpy==1.14.0
2 pip install scipy==1.0.0
```

Aby zainstalować najnowsze wersje tych bibliotek

Listing 2.2. Instalacja wymaganych modułów z najnowszej wersji

```
pip install -U numpy
pip install -U scipy
```

2.4.2. Natural Language Toolkit

NLTK jest narzędziem wspomagającym pisanie programów do przetwarzania języka naturalnego. Biblioteka dostarcza wiele istotnych dodatków, takich jak klasyfikacja, tokenizacja czy stemming (szczegółowy opis tych operacji zaprezentowane w podpunkcie 2.5).

NLTK jest projektem otwartym (ang. open-source). Jego instalacja przebiega w taki sam sposób co instalacja Scikit-learn. Należy uruchomić wiersz poleceń lub Terminal i wykorzystać do tego programu "pip".

Listing 2.3. Instalacja biblioteki NLTK z najnowszej wersji

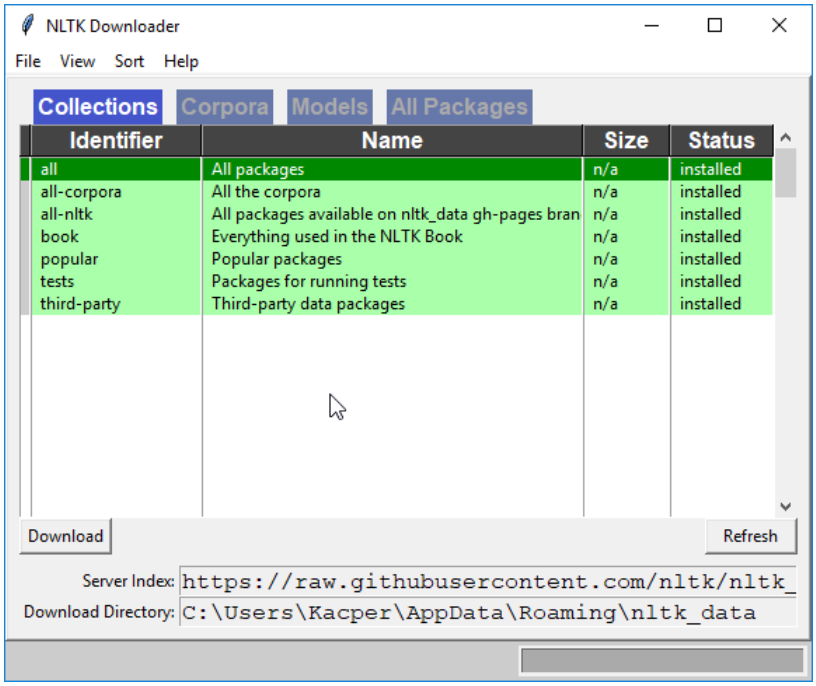
```
pip install -U nltk
```

Przetestowanie instalacji tej biblioteki należy wykonać poprzez uruchomienie programu "python" z wiersza poleceń lub Terminala w środowisku Visual Studio Code, po czym uruchomić polecenia (2.4)

Listing 2.4. Weryfikacja poprawności instalacji biblioteki NLTK

```
import nltk
nltk.download()
```

Wywołanie metody "download()" z pakietu "nltk" spowoduje pojawienie się menadżera ze wszystkimi dodatkami tej biblioteki, "NLTK Downloader", dzięki któremu w łatwy sposób można zainstalować potrzebne dodatki.



Rys. 2.5. NLTK Downloader

2.5. Przygotowanie dokumentów tekstowych

Zbyt duża różnorodność w dokumentach tekstowych może sprawić więcej problemów niż początkowo sądzimy. Długie oraz zbędne słowa czy znaki są powszechne niemal w każdych dokumentach tekstowych. Dlatego niezbędne jest wstępne przetworzenie tekstu na potrzeby klasteryzacji, tak by działała jak najkorzystniej. Przygotowywanie dokumentów można podzielić na kilka etapów, które przedstawiono w kolejnych podpunktach.

2.5.1. Tokenizacja

Tokenizacja jest to transformacja dokumentu tekstowego w taki sposób, aby rozdzielić go na pojedyncze słowa. Zazwyczaj w tym momencie używana jest tokenizacja po białych znakach (ang. whitespace). Na zadanym dokumencie tekstowym jest wykonywana operacja dzielenia tekstu w miejscach spacji, nowych linii czy tabulatorów.

Uruchomienie skryptu z listingu 2.5 podzieli nasz tekst w długą listę słów oraz wypisze 100 pierwszych.

Listing 2.5. Dzielenie tekstu po białych znakach

```
1 text = 'Python is an interpreted high-level programming language for\n general-
      purpose programming. Created by Guido van Rossum and\n first released in 1991,
      Python has a design philosophy that emphasizes code readability, and a syntax
      that allows programmers to express\n concepts in fewer lines of code,[25][26]
      notably using significant whitespace.'
2 words = text.split()
3 print(words[:100])
4 ['Python', 'is', 'an', 'interpreted', 'high-level', 'programming', 'language', 'for
      ', 'general-purpose', 'programming.', 'Created', 'by', 'Guido', 'van', 'Rossum'
      , 'and', 'first', 'released', 'in', '1991,', 'Python', 'has', 'a', 'design', '
      philosophy', 'that', 'emphasizes', 'code', 'readability,', 'and', 'a', 'syntax'
      , 'that', 'allows', 'programmers', 'to', 'express', 'concepts', 'in', 'fewer',
      'lines', 'of', 'code,[25][26]', 'notably', 'using', 'significant', 'whitespace.
      ']
```

Drugi sposób na podzielenie tekstu na części, to skorzystanie z wcześniej omówionej biblioteki Natural Language Toolkit. Biblioteka ta udostępnia gotową metodę do tokenizacji dokumentów tekstowych.

Listing 2.6. Dzielenie tekstu z wykorzystaniem biblioteki NLTK

```
1 text = 'Python is an interpreted high-level programming language for\n general-
      purpose programming. Created by Guido van Rossum and\n first released in 1991,
      Python has a design philosophy that emphasizes code readability, and a syntax
      that allows programmers to express\n concepts in fewer lines of code,[25][26]
      notably using significant whitespace.'
2 from nltk.tokenize import word_tokenize
3 tokens = word_tokenize(text)
4 print(tokens[:100])
5
6 ['Python', 'is', 'an', 'interpreted', 'high-level', 'programming', 'language', 'for
      ', 'general-purpose', 'programming', '.', 'Created', 'by', 'Guido', 'van', '
      Rossum', 'and', 'first', 'released', 'in', '1991', ',', 'Python', 'has', 'a', '
      design', 'philosophy', 'that', 'emphasizes', 'code', 'readability', ',', 'and',
       'a', 'syntax', 'that', 'allows', 'programmers', 'to', 'express', 'concepts', '
      in', 'fewer', 'lines', 'of', 'code', ',', '[', '25', ']', '[', '26', ']', '
      notably', 'using', 'significant', 'whitespace', '.']
```

Różnice pomiędzy dwoma sposobami tokenizacji są niewielkie. Drugi sposób, który wykorzystuje bibliotekę NLTK, rozdzielił również słowa od znaków interpunkcyjnych.

2.5.2. Czyszczenie

Czyszczenie dokumentów tekstowych polega na usunięciu niepotrzebnych elementów dokumentu tekstowego. Takimi elementami są: znaki interpunkcyjne, liczby i wyrazy znajdujące się na stop liście (ang. stopwords).

Python dostarcza nam gotową listę znaków interpunkcyjnych.

Listing 2.7. Lista wszystkich znaków interpunkcyjnych

```
1 import string
2 print(string.punctuation)
3
4 !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Dzięki temu jesteśmy w stanie w łatwy sposób usunąć z tekstu wszystkie znaki interpunkcyjne. Python posiada również wbudowaną metodę "isalpha()", która zwraca wartość prawdziwą tylko wtedy, kiedy wszystkie znaki w tekście należą do zbioru znaków alfanumerycznych.

Listing 2.8. Czyszczenie dokumentu tekstowego wykorzystując tokeny z 2.6

```
1 table = text.maketrans('', '', string.punctuation)
2 stripped = [w.translate(table) for w in tokens]
3 print(stripped[:100])
4 ['Python', 'is', 'an', 'interpreted', 'highlevel', 'programming', 'language', 'for'
      , 'generalpurpose', 'programming', '', 'Created', 'by', 'Guido', 'van', 'Rossum
      ', 'and', 'first', 'released', 'in', '1991', '', 'Python', 'has', 'a', 'design'
      , 'philosophy', 'that', 'emphasizes', 'code', 'readability', '', 'and', 'a', '
      syntax', 'that', 'allows', 'programmers', 'to', 'express', 'concepts', 'in', '
      fewer', 'lines', 'of', 'code', '', '', '25', '', '', '26', '', 'notably', '
      using', 'significant', 'whitespace', '']
5 words = [word for word in stripped if word.isalpha()]
6 print(words[:100])
7 ['Python', 'is', 'an', 'interpreted', 'highlevel', 'programming', 'language', 'for'
      , 'generalpurpose', 'programming', 'Created', 'by', 'Guido', 'van', 'Rossum', '
      and', 'first', 'released', 'in', 'Python', 'has', 'a', 'design', 'philosophy',
      'that', 'emphasizes', 'code', 'readability', 'and', 'a', 'syntax', 'that', '
      allows', 'programmers', 'to', 'express', 'concepts', 'in', 'fewer', 'lines', '
      of', 'code', 'notably', 'using', 'significant', 'whitespace']
```

Czyszczenie dokumentu tekstowego z elementów znajdujących się na stop liście (ang. stopwords), również można wykonać za pomocą biblioteki NLTK. Dostarcza ona predefiniowane stop listy w różnych językach.

Listing 2.9. Kasowanie słów zawartych na stop liście

```
1 from nltk.corpus import stopwords
2 words = [word for word in words if word not in stopwords.words('english')]
3 print(words[:100])
4 ['Python', 'interpreted', 'highlevel', 'programming', 'language', 'generalpurpose',
      'programming', 'Created', 'Guido', 'van', 'Rossum', 'first', 'released', '
     Python', 'design', 'philosophy', 'emphasizes', 'code', 'readability', 'syntax',
      'allows', 'programmers', 'express', 'concepts', 'fewer', 'lines', 'code', '
     notably', 'using', 'significant', 'whitespace']
```

Istnieje również możliwość wygenerowania własnej stop listy na podstawie analizowanych dokumentów tekstowych. Zwykle taką listę generuje się zliczając ilość unikalnych słów we wszystkich dokumentach tekstowych, a następnie uznając te najbardziej popularne za słowa, które nie są istotne do klasyfikacji dokumentów. Ważnym elementem w tym procesie, jest wyznaczenie progu ilości wystąpień słowa, od którego zależy czy jest ono istotne czy nie.

2.5.3. Normalizacja

Normalizację (ang. normalization) konkretnych słów w dokumencie tekstowym można wykonać na wiele różnych sposobów. Zwykle polega ona na przyjęciu pewnego wzorca normalizacji dla całego dokumentu i zastosowaniu go do wszystkich słów w tekście. Jedną z metod normalizacji jest zwykła zamiana wszystkich słów w dokumencie na słowa z małymi literami (ang. lowercase) [4]. Kolejnym krokiem jest normalizacja dat. Zapis dat w różnych częściach świata jest różny [5]. Jednak każdy miesiąc posiada swój unikalny skrót [6], który należałoby znormalizować, czyli zamienić na pełne nazwy miesięcy.

Ostatnim etapem normalizacji jest stemming. Pierwsze próby zastosowania algorytmu stemmingu zostały zaproponowane w latach 60. XX wieku. Słowa o takim samym znaczeniu, które zapisane są na kilka różnych sposobów mogą być poddane stemmingowi. Proces ten kasuje końcówki fleksyjne ze słów w dokumencie tekstowym pozostawiając tylko część słowa zwaną rdzeniem. Można wyróżnić kilka rodzajów algorytmów stemmingu:

- Lancaster Stemmer,
- Snowball Stemmer,

- Porter Stemmer.

Listing 2.10. Zastosowanie powyższych rodzajów stemmingu

```
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem import SnowballStemmer
from nltk.stem.porter import PorterStemmer
wordsToStem = ['multiply', 'connecting', 'connected', 'said', 'provision', '
    traditional', 'plotted', 'caresses']
stemWords = [LancasterStemmer().stem(word) for word in wordsToStem]
print(stemWords)
stemWords = [SnowballStemmer('english').stem(word) for word in wordsToStem]
print(stemWords)
stemWords = [PorterStemmer().stem(word) for word in wordsToStem]
print(stemWords)

['multiply', 'connect', 'connect', 'said', 'provid', 'tradit', 'plot', 'caress']
['multipli', 'connect', 'connect', 'said', 'provis', 'tradit', 'plot', 'caress']
['multipli', 'connect', 'connect', 'said', 'provis', 'tradit', 'plot', 'caress']
```

2.5.4. TF-IDF

Ważnym elementem grupowania dokumentów tekstowych, jest dobór odpowiednich cech opisujących dokument. Cechy ekstrachowane są z dokumentów tekstowych w ten sposób, aby dokumenty o podobnej treści uzyskiwały podobne wektory wartości cech. Dokumenty tekstowe o treściach różniących się od siebie powinny zostać opisane różnymi wektorami wartości cech.

$TF - IDF$ (ang. TF - term frequency, IDF - inverse document frequency) jest klasyczną metodą, dzięki której możliwe jest przypisywanie wag poszczególnym słowom kluczowym opisującym dokument tekstowy.

$$\begin{bmatrix} w_{d_1t_1} & . & . & w_{d_nt_1} \\ . & . & & . \\ . & & . & . \\ w_{d_1t_n} & . & . & w_{d_nt_n} \end{bmatrix} \tag{2.1}$$

gdzie $w_{d_1t_1}$ to obliczona waga słowa t_1 w dokumencie d_1.

Aby wyznaczyć wagi dla dokumentu tekstowego i policzyć współczynnik $TF - IDF$ należy w pierwszej kolejności obliczyć miary:

- częstotliwości występowania wyrażeń (ang. term frequency - TF) - $TF(t_i, d)$ - liczba wystąpień słowa t_i w dokumencie d,
- częstotliwości dokumentów (ang. document frequency) - $DF(t_i)$ - liczba dokumentów tekstowych, w których istnieje słowo t_i,
- odwrotnej częstości dokumentu (ang. inverse document frequency - IDF) - $IDF(t_i) = 1 + \ln\left(\frac{|D|}{DF(t_i)}\right)$ [7] gdzie |D| to liczba wszystkich dokumentów tekstowych,

$TF - IDF$ jest iloczynem miar TF oraz IDF. Przykładowo dla podanych poniżej trzech dokumentów tekstowych:

- $d1$ = The game of life is a game of everlasting learning,
- $d2$ = The unexamined life is not worth living,
- $d3$ = Never stop learning

należy na samym początku policzyć "term frequency", czyli ilość występowania danego słowa w dokumencie tekstowym. Drugą istotną rzeczą w tym etapie jest znormalizowanie TF. Normalizacja odbywa się za pomocą wzoru:

$$NTF(t_i, d) = \frac{TF(t_i, d)}{|C_d|} \tag{2.2}$$

gdzie $|C_d|$ to ilość słów w dokumencie tekstowym.

Tab. 2.1. Obliczone wartości $TF(t_i, d1)$ i $NTF(t_i, d1)$

$d1$	the	game	of	life	is	a	everlasting	learning
$TF(t_i, d1)$	1	2	2	1	1	1	1	1
$NTF(t_i, d1)$	0.1	0.2	0.2	0.1	0.1	0.1	0.1	0.1

Tab. 2.2. Obliczone wartości $TF(t_i, d2)$ i $NTF(t_i, d2)$

$d2$	the	unexamined	life	is	not	worth	living
$TF(t_i, d2)$	1	1	1	1	1	1	1
$NTF(t_i, d2)$	0.1428	0.1428	0.1428	0.1428	0.1428	0.1428	0.1428

Tab. 2.3. Obliczone wartości $TF(t_i, d3)$ i $NTF(t_i, d3)$

$d3$	never	stop	learning
$TF(t_i, d3)$	1	1	1
$NTF(t_i, d3)$	0.3333	0.3333	0.3333

Kolejnym krokiem jest policzenie współczynnika IDF dla każdego termu ze wszystkich dokumentów tekstowych. Ilość wszystkich dokumentów jest równa 3, stąd

$$|D| = 3 \tag{2.3}$$

Korzystając ze wzoru

$$IDF(t_i) = 1 + \ln\left(\frac{|D|}{DF(t_i)}\right) \tag{2.4}$$

Tab. 2.4. Wartości współczynnika IDF dla wszystkich termów

Termy	IDF
the	1.40546
game	2.09861
of	2.09861
life	1.40546
is	1.40546
a	2.09861
everlasting	2.09861
learning	1.40546
unexamined	2.09861
not	2.09861
worth	2.09861
living	2.09861
never	2.09861
stop	2.09861

Ostatnim krokiem jest policzenie dla wszystkich termów współczynników $TF - IDF$. Można zauważyć, że czym większa obliczona waga, tym większa istotność danego słowa w dokumencie, dlatego też uważa się, że termy, które mają największą wagę w danym dokumencie tekstowym, są tymi cechami, które dosyć dobrze opisują jego tematykę. Zaletą współczynnika $TF-IDF$ jest to, że przy odróżnianiu jednego dokumentu tekstowego od innych, wpływ ma nie tylko częstość występowania w nim danego słowa, ale też występowanie tego słowa w innych dokumentach.

Tab. 2.5. Wartości współczynnika $TF - IDF$ dla dokumentów tekstowych

Termy	$d1$	$d2$	$d3$
the	0.140546	0.200699	0
game	0.419722	0	0
of	0.419722	0	0
life	0.140546	0.200699	0
is	0.140546	0.200699	0
a	0.209861	0	0
everlasting	0.209861	0	0
learning	0.140546	0	0.468439
unexamined	0	0.299681	0
not	0	0.299681	0
worth	0	0.299681	0
living	0	0.299681	0
never	0	0	0.699466
stop	0	0	0.699466

Policzenie $TF - IDF$ za pomocą Pythona sprowadza się do wykorzystania biblioteki Scikit-learn posiadającej w sobie klasę TfidfVectorizer, która na wejściu otrzymuje listę dokumentów, a następnie zwraca wszystkie wagi słów. Klasa ta posiada również dużo parametrów, jak "stopwords='english'", która wyklucza wszystkie słowa zawierające się na stop liście.

Listing 2.11. Przykład kodu wykorzystanego do policzenia $TF - IDF$ dla dokumentów tekstowych

```
1  import numpy as np
2  from sklearn.feature_extraction.text import TfidfVectorizer
3  def display_scores(vectorizer, tfidf_result):
4      scores = zip(vectorizer.get_feature_names(), np.asarray(tfidf_result.sum(axis
           =0)).ravel())
5      sorted_scores = sorted(scores, key=lambda x: x[1], reverse=True)
6      allTerms = ["{0} ({1})".format(z[0], z[1]) for z in sorted_scores]
7      for term in allTerms:
8          print(term)
9  text1 = "The game of life is a game of everlasting learning"
10 text2 = "The unexamined life is not worth living"
11 text3 = "Never stop learning"
```

```
tfidf = TfidfVectorizer()
tfidfMatrix = tfidf.fit_transform([text1, text2, text3])
display_scores(tfidf, tfidfMatrix)

learning (0.699736372829852)
never (0.6227660078332259)
stop (0.6227660078332259)
game (0.5946064659329678)
of (0.5946064659329678)
is (0.5436769522249277)
life (0.5436769522249277)
the (0.5436769522249277)
living (0.4175666238781924)
not (0.4175666238781924)
unexamined (0.4175666238781924)
worth (0.4175666238781924)
everlasting (0.2973032329664839)
```

2.6. Klasteryzacja danych

2.6.1. Algorytm k-means

Algorytm k-Means jest wykorzystywany do partycjonowania danych obserwacyjnych w predefiniowaną ilość klastrów. Algorytm opracowany przez J MacQueen-a (patrz [8]) rozpoczyna się od losowego rozmieszczenia zestawu środków klastrów (μ) uważanych za środki klastrów. Podczas każdego kroku aktualizacji wszystkie obserwacje x są przypisywane do najbliższego punktu środkowego (patrz równanie 2.5). W standardowym algorytmie każda obserwacja x (punkt z danymi) jest przypisana do jednego klastra. Jeśli kilka środków klastrów ma taką samą odległość do obserwacji x, wtedy punkt jest przypisywany losowo do wybranego klastra z puli równoodległych klastrów.

Przydzielenie jednej obserwacji do klastra, którego średnia ma najmniejszą kwadratową odległość euklidesową.

$$S_i = \left\{ x_p : \left\| x_p - \mu_i \right\|^2 \leqslant \left\| x_p - \mu_j \right\|^2 \ \forall j, 1 \leqslant j \leqslant k \right\} \tag{2.5}$$

gdzie: x_p to wektor cech opisujący dokument przypisany dokładnie do jednego klastra S_i, μ_i to środek klastra, μ_j to środek innego klastra.

Następnie środki klastrów są aktualizowane poprzez obliczenie średniej ze wszystkich obserwacji przypisanych do danego klastra (2.6):

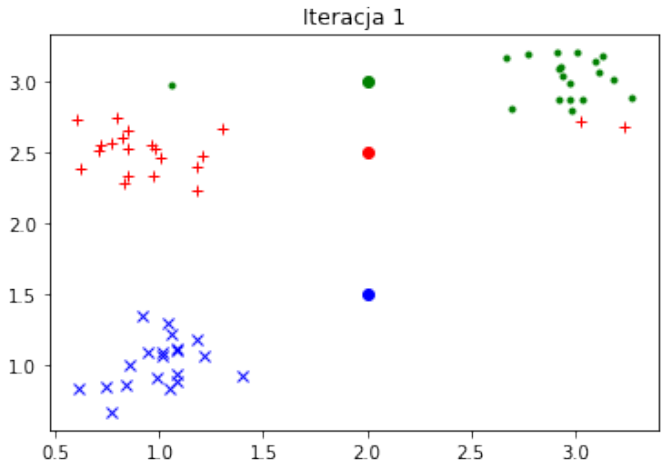
$$\mu_i = \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j \tag{2.6}$$

gdzie: μ_i to środek klastra, $|S_i|$ wszystkie obserwacje przypisane do klastra i, x_j wektor cech opisujący dokument należący do S_i.

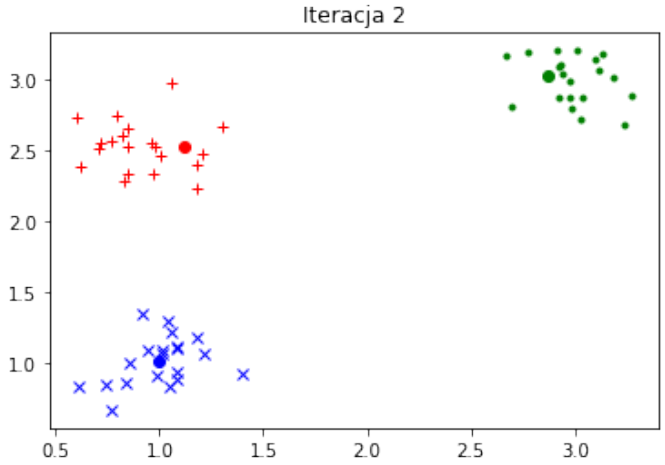
Proces aktualizacji środków klastrów powtarza się do momentu, aż wszystkie obserwacje przestają zmieniać przynależność do klastrów. Na zaprezentowanych poniżej rysunkach, przedstawiono przebieg przykładowego procesu klasteryzacji. W pierwszej iteracji położenie środków klastrów jest określane losowo, a następnie wszystkie punkty danych są przypisywane do najbliższych klastrów.

W drugiej iteracji położenie środków klastrów jest aktualizowane na podstawie przypisanych do klastra obserwacji. Nowe środki obliczamy, jako średnia wszystkich obserwacji przydzielonych do danego klastra. Ostatnią iteracją jest kolejna aktu-

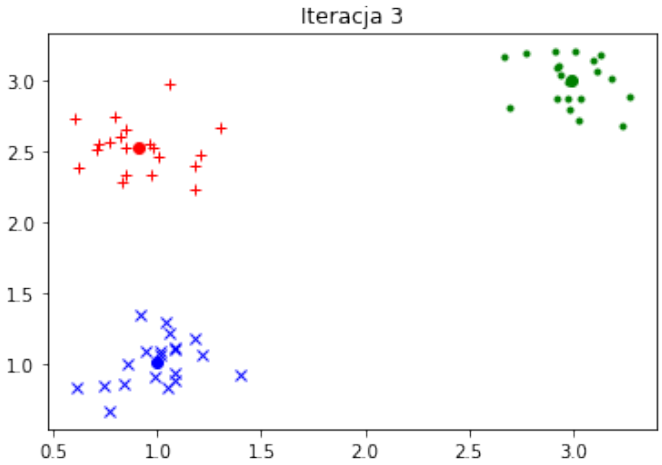
alizacja środków klastrów. Ponieważ w tej iteracji nie zmieniła się przynależność obserwacji do klastrów, to algorytm na tym etapie kończy swoje działanie.



Rys. 2.6. Algorytm k-Means: Inicjalizacja algorytmu poprzez losowe rozmieszczenie środków klastrów i wyznaczenie przynależność punktów danych do klastrów

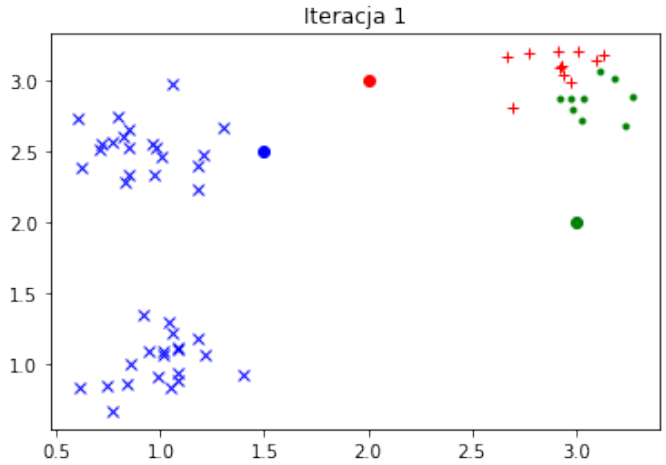


Rys. 2.7. Algorytm k-Means: druga iteracja - aktualizacja pozycji środków klastrów oraz ponowne przydzielenie obserwacji do klastrów

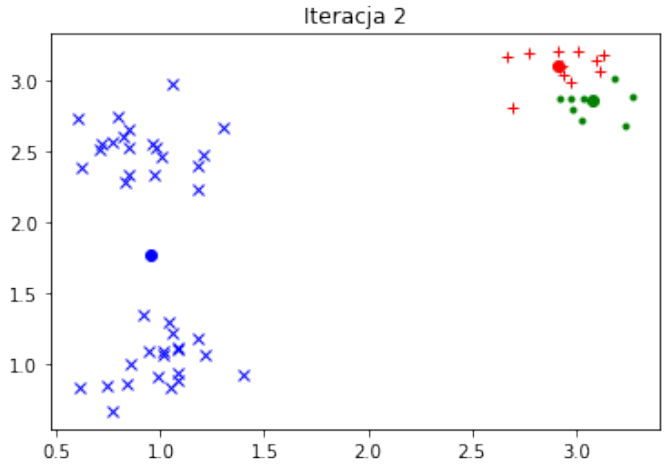


Rys. 2.8. k-Means: trzecia iteracja - aktualizacja pozycji środków klastrów

Głównym problemem algorytmu k-Means jest jego zależność od początkowo wybranych pozycji środków klastrów. Dwa centra mogą znaleźć tą samą grupę i podzielić się obserwacjami, w momencie kiedy dla dwóch środków klastrów przydzieli się punkty danych z tej samej grupy. Kolejne iteracje będą powielały błąd stworzony na samym początku działania algorytmu. Błędne początkowe położenie środków klastrów pokazano na rysunku 2.9.



Rys. 2.9. Algorytm k-Means: Złe początkowe pozycje środków klastrów



Rys. 2.10. Algorytm k-Means: Aktualizacja środków klastrów oraz ponowne przydzielenie punktów danych

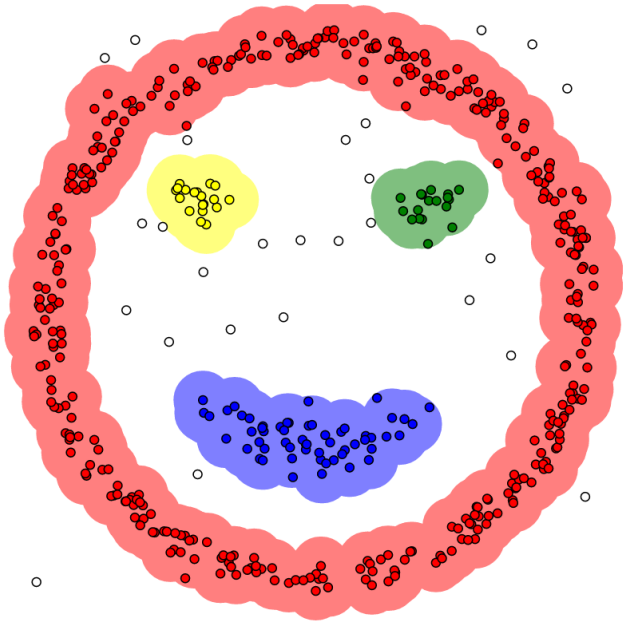
k-Means najlepiej radzi sobie z klasteryzacją danych w których, klastry są mniej więcej równej wielkości i mają kolisty kształt.

2.6.2. DBSCAN

Algorytm DBSCAN (ang. Density Based Spatial Clustering of Application with Noise) jest jednym z algorytmów typu gęstościowego. Został zaprezentowany w 1996 roku przez czterech naukowców: Martin Ester, Hans-Peter Kriegel, Jörg Sander oraz Xiaowei Xu [9]. Działanie algorytmu wymaga podania dwóch parametrów: maksymalny promień sąsiedztwa *Eps* i minimalną ilość punktów oznaczaną jako *minPts*. Algorytm rozpoczyna działanie od wybrania dowolnego punktu w zbiorze danych. Jeśli w obrębie *Eps* znajduje się minimalna ilość punktów *minPts* w odległości od punktu początkowego to znalezione punkty są przydzielane do aktualnego klastra.

Algorytm kontynuuje rozbudowywanie klastra od wcześniej przydzielonych punktów. Rozbudowywanie klastra dobiega końca w momencie kiedy nie istnieje minimalna ilość *minPts* punktów w obrębie *Eps* od najbardziej wysuniętego punktu w klastrze.

W przeciwieństwie do algorytmu k-Means, algorytm DBSCAN potrafi dokonać klasteryzacji danych tworzących skupiska o różnych kształtach i wielkościach. Ponadto algorytm potrafi klasteryzować zbiory danych, które zawierają się w innej grupie co ilustruje rysunek 2.11

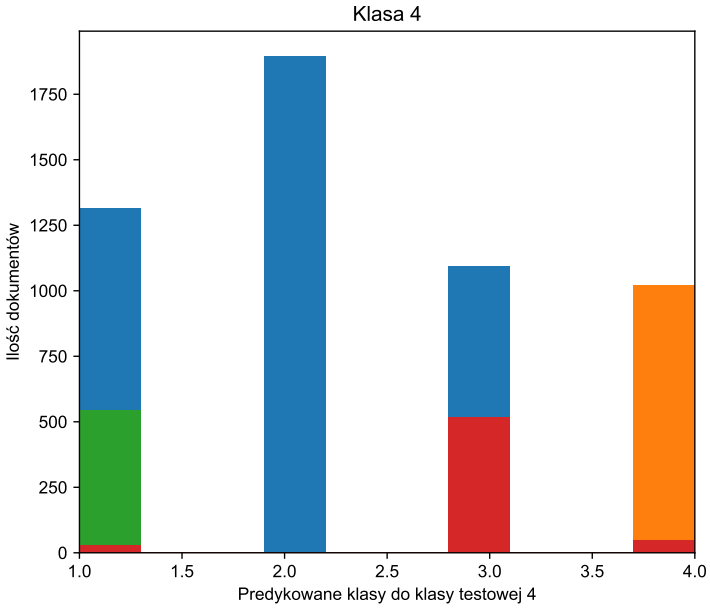


Rys. 2.11. Przynależność punktów do klastrów w algorytmie DBSCAN

2.7. Weryfikacja danych

2.7.1. Histogram

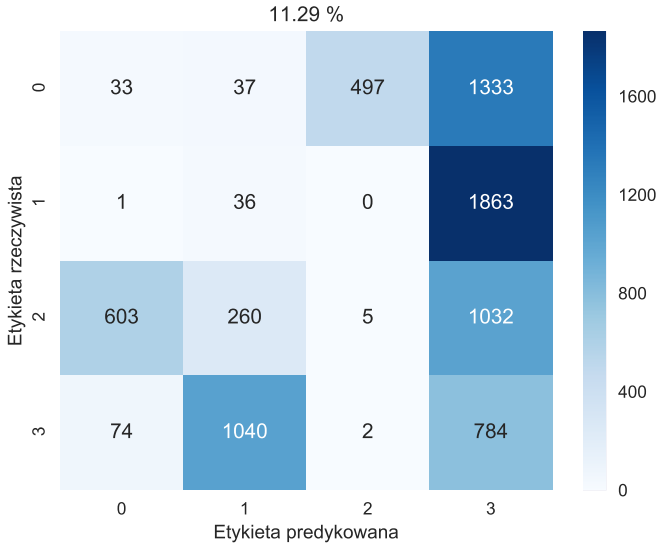
Histogram jest jednym ze sposobów przedstawienia rozkładu empirycznego. Posiada on oś x, na której są położone słupki pewnych grup lub cech z danego zbioru danych. Oś y reprezentuje ilość wystąpień danej grupy lub cechy. Histogram można wykorzystać na przykład do prezentacji ilości wystąpień słów we wszystkich dokumentach tekstowych z jednej grupy. Dzięki takiemu przedstawieniu można łatwo określić jaka grupa tych dokumentów posiada jaką tematykę. W niniejszej pracy histogram jest wykorzystywany, aby określić ilość wystąpień klas zwróconych przez algorytm klasteryzacji, względem klas rzeczywistych tych samych dokumentów tekstowych.



Rys. 2.12. Przykładowy histogram

2.7.2. Macierz pomyłek

Macierz pomyłek jest narzędziem stosowanym do oceny jakości wyników z algorytmu klasteryzacji. Na osi x znajdują się etykiety klas, które wyznaczane są przez algorytm klasteryzacji, a na osi y rzeczywiste etykiety klas.



Rys. 2.13. Przykładowa macierz pomyłek

Odczytanie danych z macierzy pomyłek z 2.13 następuje w taki sposób, że każdy wiersz to klasa rzeczywista analizowanych dokumentów tekstowych. Kolumny oznaczają klasy, które wyznaczył algorytm klasteryzacji. Z macierzy przedstawionej na rysunku 2.13 możemy odczytać, że dla 33 dokumentów tekstowych mających przypisaną etykietę rzeczywistą "0", algorytm również znalazł oraz przypisał taką samą etykietę.

2.7.3. Kombinacje klas predykowanych we wszystkich możliwościach

Wygenerowanie wszystkich kombinacji przypisań klas predykowanych, pozwoli na znalezienie najlepszego mapowania do klas rzeczywistych. Algorytmy klasteryzacji należą do technik uczenia nienadzorowanego. Z tego względu podczas wielokrotnego uruchamiania algorytmu, pogrupowane dokumenty tekstowe mogą raz znaleźć się w klastrze o etykiecie "1", a innym razem w klastrze "4". Na to jakie algorytm klasteryzacji przypisze etykiety dokumentom tekstowym, nie mamy wpływu dlatego aby zweryfikować dokładność klasteryzacji potrzebne jest dopasowanie etykiet przypisanych przez algorytm do etykiet rzeczywistych.

W tym celu stosuje się zamiany klas wygenerowanych przez algorytm w różnych kombinacjach. Schemat działania zamiany klas wygląda w taki sposób, że należy wziąć wyjściową listę etykiet z algorytmu (na 2.6 jest to 1 kombinacja), utworzyć macierz pomyłek oraz policzyć trafność przypisania klas (ang. accuracy). Następnie przesunąć każdą predykowaną klasę o jedną klasę w przód, aż do momentu uzyskania początkowych wyników. Dla każdego przypadku liczymy trafność przypisania klas. Wybieramy takie przypisanie, w którym trafność w skali procentowej jest największa.

Tab. 2.6. Przykładowe kombinacje dla etykiet predykowanych

1 kombinacja	2 kombinacja	3 kombinacja	4 kombinacja
4	1	2	3
4	1	2	3
2	3	4	1
4	1	2	3
1	2	3	4
3	4	1	2
3	4	1	2
2	3	4	1
3	4	1	2
4	1	2	3

Rozdział 3

Opis i wyniki przeprowadzonych testów

3.1. Przebieg procesu testowania

Część praktyczna pracy polegała na implementacji programu do klasteryzacji dokumentów tekstowych oraz przeprowadzeniu testów weryfikujących skuteczność opracowanego rozwiązania. Opracowany system na wejściu otrzymuje listę dokumentów tekstowych w postaci pliku ".csv", a w rezultacie tworzy nowy plik ".csv" z listą dokumentów tekstowych oraz przypisanymi do nich etykietami klastrów. Przygotowanie takiego projektu składało się z kilku etapów:

- znalezienie dużego zbioru dokumentów tekstowych potrzebnego do przeprowadzenia testów weryfikujących skuteczność systemu,
- przetworzenie dokumentów tekstowych na format akceptowalny przez procedury wykorzystane do implementacji klasteryzacji,
- implementacja systemu do wstępnego przetwarzania i klasteryzacji dokumentów tekstowych,
- przeprowadzenie testów weryfikujących skuteczność wybranej metody klasteryzacji.

Celem eksperymentu jest znalezienie jak najbardziej podobnych do siebie dokumentów, korzystając tylko z metod przetwarzania dokumentów tekstowych oraz klasteryzacji. W pracy zostały wykorzystane takie metody jak, "stemming", "tf-idf".

Problem polega na tym, że klasteryzacja, inaczej zwana klasyfikacją nienadzorowaną, zwraca podzielone dokumenty na grupy w różny sposób. Interpretacja wygenerowanych klastrów wymaga zastosowania kilku metod w celu weryfikacji czy dany wynik jest poprawny bądź nie. Dokumenty tekstowe znajdujące się w zbiorze testowym posiadają etykiety opisujące ich przynależność do predefiniowanych klas. Zatem weryfikacja skuteczności klasteryzacji będzie polegała na porównaniu dla każdego dokumentu jego klasy referencyjnej z klasą przypisaną mu przez algorytm klasteryzacji.

3.2. Wykorzystane dane do przeprowadzenia eksperymentów

Do przeprowadzenia eksperymentów został wykorzystany zbiór danych znalezionych na jednej ze stron internetowych [10] o nazwie roboczej `"ag_news_csv"`. Dokumenty tekstowe zawarte w tym zbiorze są podzielone na 4 grupy.

Tab. 3.1. Klasy z danego zbioru danych

Numer klasy	1	2	3	4
Nazwa klasy	World	Sports	Business	Sci/Tech

Zbiór danych zawiera w sobie dwa pliki `"test.csv"` oraz `"train.csv"`. Ilość dokumentów dla obu plików jest różna i prezentuje się w taki sposób:

Tab. 3.2. Ilość dokumentów tekstowych w poszczególnych klasach i zbiorach danych

Numer klasy	test.csv	train.csv
1	1900	30000
2	1900	30000
3	1900	30000
4	1900	30000

Struktura w pliku z danymi źródłowymi jest taka, że każda nowa linia to osobny dokument tekstowy. Informacje rozdzielone są separatorem przecinka. W pliku są trzy informacje, takie jak numer klasy oryginalnej, nazwa artykułu oraz treść artykułu.

Powodem dla którego został wybrany ten zbiór danych, jest fakt, że zbiór ten nie posiada dużo klas, dzięki czemu stopień trudności klasteryzacji jest mniejszy. Wydobycie cech z dokumentów tekstowych w podanym zbiorze, możliwe jest dzięki odpowiedniej długości słów w nich zawartych.

W wypadku, gdy dokument tekstowy będzie zbyt krótki, po całym procesie przetwarzania może nie mieć ani jednej cechy opisującej ten dokument. Cechami w tym przypadku są częstości występowania słów kluczowych, których lista ustalana jest w wyniku przetwarzania wstępnego wszystkich tekstów.

Podczas analizowania treści tych dokumentów, napotkałem na wiele różnych elementów, które nie wnoszą nic do ich istotności. Są to m.in. adresy URL, akronimy, czy elementy znaczników HTML, ponieważ dokumenty tekstowe w podanym zbiorze pochodzą ze stron internetowych, dlatego w kilku przypadkach parsowanie i kasowanie HTML nie zadziałało poprawnie. Takie elementy mogą zakłócić działanie algorytmu. Bardzo ważny jest moment przetwarzania dokumentów tekstowych, tak by zostały w nim tylko istotne słowa.

3.3. Cele szczegółowe

3.3.1. Parsowanie HTML

Niektóre dokumenty z podanego zbioru posiadają w sobie różnego rodzaju znaczniki HTML czy nawet adresy URL, które są zbędnymi słowami jeśli chodzi o klasteryzację. W tym celu została napisana metoda, która przyjmuje treść dokumentu tekstowego jako parametr i zwraca ten sam dokument bez znaczników i znaków specjalnych takich jak "<". Metody te korzystają z bibliotek "html" oraz "re" dostarczonych przez język Python.

Listing 3.1. Metody do kasowania znaczników HTML oraz adresów URL

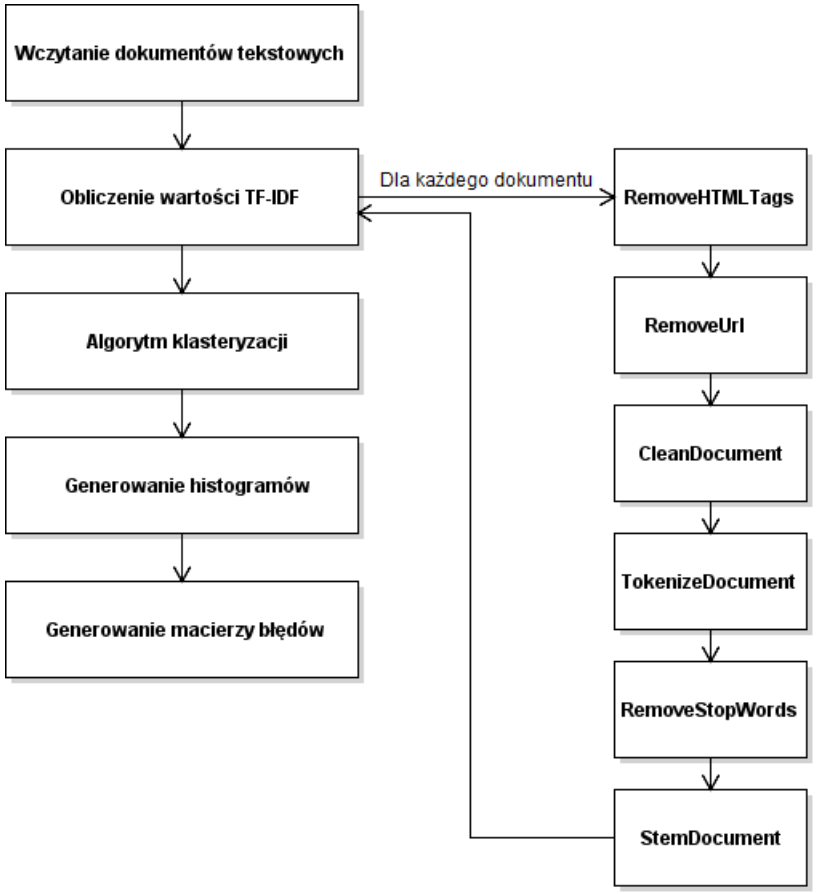
```
1  import html
2  import re
3
4  def RemoveHTMLTags(document):
5      unescapedDocument = html.unescape(document)
6      return re.sub('<[^<]+?>', '', unescapedDocument.lower())
7
8  def RemoveUrl(document):
9      return re.sub(r'(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w\.-]*)*\/?\S',
           '', document.lower(), flags=re.MULTILINE)
10
11 documentWithHtmlTags = "&lt;A HREF=""http://www.reuters.co.uk/financeQuoteLookup.
       jhtml?ticker=6753.T qtype=sym infotype=info qcat=news""&gt;6753.T&lt;/A&gt;"
12 documentWithUrl = "USATODAY.com - The federal government "
13
14 documentWithoutHtmlTags = RemoveHTMLTags(documentWithHtmlTags)
15 print(documentWithoutHtmlTags)
16 #6753.t
17
18 documentWithoutUrls = RemoveUrl(documentWithUrl)
19 print(documentWithoutUrls)
20 # - the federal government
```

Obie metody można wykorzystać w funkcji "PrepareDocument", która została zaprezentowana w rozdziale 3.3.

Metoda "RemoveHTMLTags" przyjmuje jako parametr dokument tekstowy. W pierwszej kolejności korzystając z metody "html.unescape", która zamienia wszystkie znaki specjalne (np. >, >), na ich odpowiedniki w formacie Unicode, a następnie poprzez użycie wyrażeń regularnych usuwa znaczniki HTML, bez kasowania wartości danego znacznika. Metoda "RemoveUrl" również przyjmuje dokument

tekstowy jako parametr. Korzystając z wyrażenia regularnego na danym tekście, wszystkie znalezione wzorce są zamieniane na pusty tekst.

3.3.2. Schemat działania algorytmu



Rys. 3.1. Diagram przedstawiający etapy działania programu do klasteryzacji dokumentów tekstowych

Na samym początku należy skonfigurować algorytm, tak by działał z odpowiednimi danymi. W przypadku algorytmu k-means należy określić, na ile klas chcemy podzielić dane. Należy również określić ścieżkę dostępu do pliku `".csv"` ze zbiorem danych, oraz nazwami klas. Kolejnym etapem jest pobranie wszystkich dokumentów tekstowych do listy.

Listing 3.2. Pobieranie dokumentów tekstowych z pliku .csv

```
import csv

class RowFromFile(object):
    def __init__(self, cluster=None, articleName=None, articleContent=None):
        self.cluster= cluster
        self.articleName = articleName
        self.articleContent = articleContent

original_documents = []
with open("sciezka.csv", encoding="utf8") as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='"')
    for row in spamreader:
        if len(row) == 3:
            original_documents.append(RowFromFile(int(row[0]), row[1], row[2]))
```

Dalej, wszystkie dokumenty są przygotowywane zgodnie z wymaganiami. Przygotowanie dokumentu tekstowego dzieli się na 4 etapy:

1. czyszczenie (patrz 2.5.2),
2. tokenizacja (patrz 2.5.1),
3. usuwanie słów zawartych w stop liście ,
4. normalizacja, czyli stemming (patrz 2.5.3).

Listing 3.3. Implementacja metod odpowiedzialnych za przygotowanie dokumentów tekstowych

```
all_documents = [x.articleName + ' ' + x.articleContent for x in original_documents
    ]
tfidf = TfidfVectorizer(tokenizer=PrepareDocument)
tfidfMatrix = tfidf.fit_transform(all_documents)

def PrepareDocument(document):
    document = CleanDocument(document)
    tokenizedDoc = TokenizeDocument(document)
    removedDoc = RemoveStopWords(tokenizedDoc)
    stemmedDoc = StemDocument(removedDoc)
    return stemmedDoc

def TokenizeDocument(document):
    tokens = [word for sent in nltk.sent_tokenize(document) for word in nltk.
        word_tokenize(sent)]
    filtered_tokens = []
    for token in tokens:
```

```
16          if re.search('[a-zA-Z]', token):
17              if len(token) >= 5:
18                  filtered_tokens.append(token)
19      return filtered_tokens
20
21  def StemDocument(tokenizedDoc):
22      return [stemmer.stem(t) for t in tokenizedDoc if t]
23
24  def CleanDocument(document):
25      document = document.replace("\\n", " ")
26      document = re.sub("[^a-zA-Z ]", " " , document)
27      return document.lower()
28
29  def RemoveStopWords(tokenizedDocument):
30      removedStopWords = [word for word in tokenizedDocument if not word in stopwords
            ]
31      return removedStopWords
```

Parametr "tokenizer" wymusza wykorzystanie naszej metody do celów tokenizacji, który dodatkowo ma zaimplementowane inne elementy przygotowania dokumentu tekstowego. Na samym końcu uruchomiamy algorytm klasteryzacji.

Listing 3.4. Uruchomienie algorytmu k-Means

```
1  from sklearn.cluster import KMeans
2  km = KMeans(n_clusters=num_clusters, max_iter=4)
3  km.fit(tfidfMatrix)
4  clusters = km.labels_.tolist()
```

Algorytm k-means 2.6.1 w bibliotece Scikit-learn zwraca listę przypisanych klas do dokumentów tekstowych w takiej samej kolejności, w jakiej otrzymał je na wejściu.

3.3.3. Weryfikacja poprawności rezultatów

Algorytm klasteryzacji ma jedną istotną wadę: numery klastrów, które zostały zwrócone, nie muszą odzwierciedlać faktycznych klas dokumentów tekstowych. Dlatego też ważnym elementem jest poprawna weryfikacja jakości wyników. W pracy opisane zostaną dwie metody weryfikacji zwróconych klas.

Dokumenty tekstowe ze zbioru danych posiadają już przypisaną klasę do każdego dokumentu. Ta wiedza jest istotna podczas generowania macierzy pomyłek 2.7.2. Weryfikacja dokładności klasyfikacji dokumentów za pomocą klasteryzacji odbywa

się kilkuetapowo:

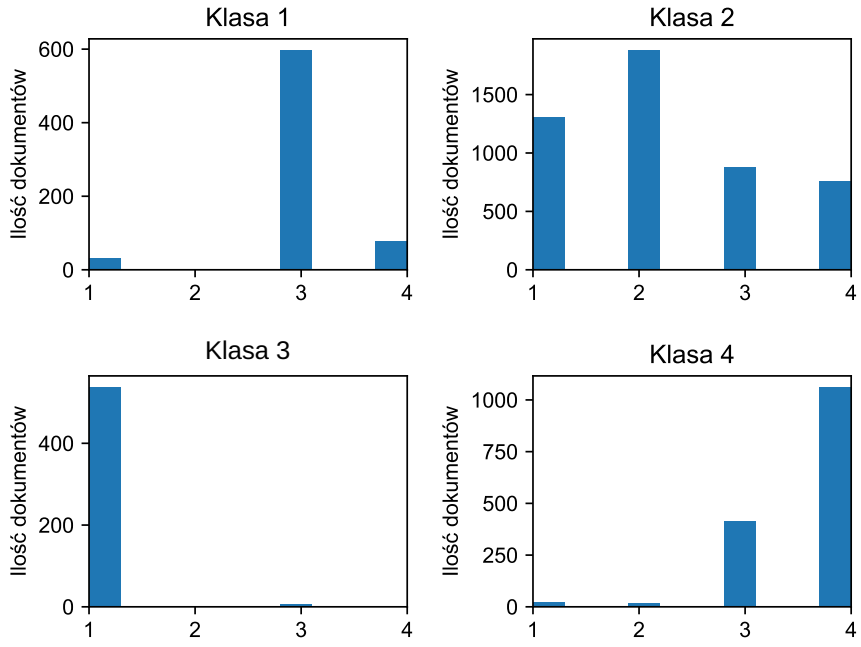
- wygenerowanie macierzy pomyłek, dla predykowanych klas wygenerowanych za pomocą algorytmu klasteryzacji,
- wygenerowanie histogramu z rozkładem klas w poszczególnych klastrach,
- zmapowanie na nowo predykowanych klas, korzystając z informacji z histogramu, oraz wygenerowanie macierzy pomyłek,
- zmapowanie predykowanych klas na wszystkie sposoby prezentacji, wygenerowanie dla każdej kombinacji macierzy pomyłek, oraz policzenie trafności przypisanych klas.

3.4. Wyniki testów

Wykorzystując przedstawione w poprzednim punkcie kroki, istnieje możliwość zbadania wyników klasteryzacji na danym zbiorze danych `"ag_news_csv"`.

Na początku zostały przedstawione wyniki z pierwszego uruchomienia algorytmu na pliku "test.csv", który zawiera po 1900 dokumentów tekstowych z każdego klastra.

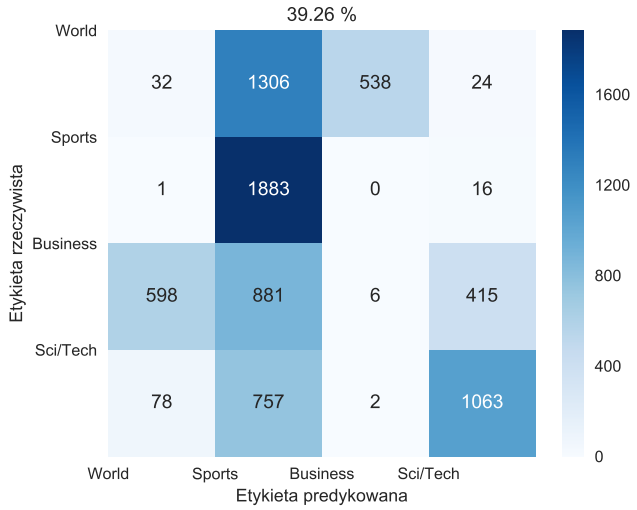
Uruchomienie programu na zbiorze testowym, wyświetli 4 wykresy w osobnych oknach. Wykresów zostały przedstawione oraz omówione na rysunkach 3.2, 3.3, 3.4 i 3.5. Jednym z wykresów jest wygenerowany histogram dla 4 klas rzeczywistych wraz z ilością etykiet znalezionych przez algorytm klasteryzacji.



Rys. 3.2. Uruchomienie 1: Histogram

Wygenerowane na powyższym rysunku 3.2 histogramy, odzwierciedlają ilość przypisanych klas przez algorytm klasteryzacji do dokumentów tekstowych należących do predefiniowanej klasy.

Poniższy rysunek 3.3 przedstawia macierz błędu, który został wygenerowany na podstawie etykiet rzeczywistych oraz predykowanych zwróconych przez algorytm klasteryzacji.



Rys. 3.3. Uruchomienie 1: Macierz błędu dla danych zwróconych z algorytmu

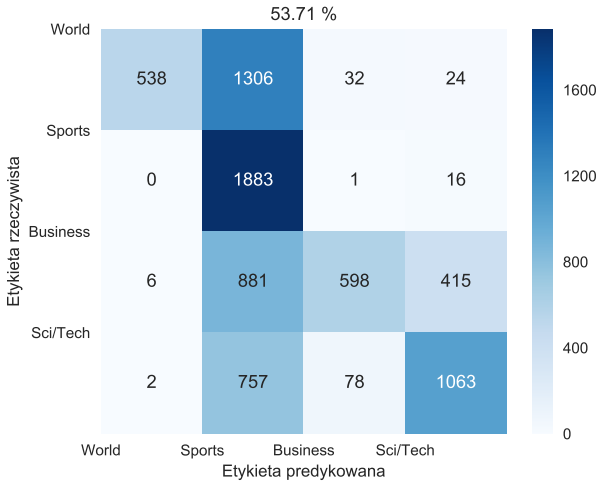
Nazwą macierzy jest policzona metryka trafności (ang. accuracy) wyników. Idealne uruchomienie posiadałoby trafność na poziomie 100 %, a macierz miałaby wszystkie dokumenty tekstowe znalezione po przekątnej. Na tym etapie wynik poprawności algorytmu klasteryzacji dla tego zbioru danych to 39.26 %.

Można jednak spróbować zrobić nowe mapowanie klas odkrytych przez algorytm, dzięki wcześniej wygenerowanemu histogramowi 3.2. Zakładając, że dla dokumentów z klasy "1" algorytm przypisał nowe etykiety, to uznać można, że klasa predykowana "3" jest naszą oryginalną "1". Tak samo postępujemy z kolejnymi histogramami. Dzięki temu można przyjąć nowe mapowanie klas.

Tab. 3.3. Uruchomienie 1: Nowe mapowanie klas predykowanych

Jaka klasa	Predykowana klasa
1	3
2	2
3	1
4	4

W przedstawionym przypadku należy zamienić wszystkie "3" na "1" oraz "1" na "3". Takie mapowanie utworzy nową listę etykiet, a macierz błędu wyglądać będzie następująco:



Rys. 3.4. Uruchomienie 1: Macierz błędu dla danych po zmapowaniu etykiet z histogramów

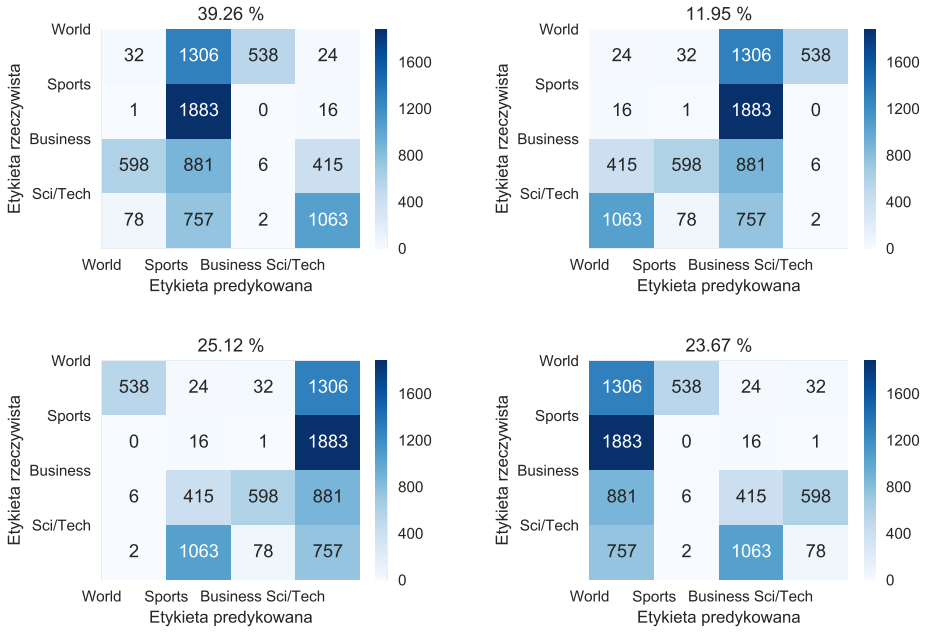
Takie mapowanie klas, zwiększyło trafność, aż do 53.71 %.

Na koniec należy skorzystać jeszcze z jednej opcji weryfikacji poprawności danych. Zamiana predykowanych klas w różnych kombinacjach przedstawiona została w 2.7.3. Mapowanie w ten sposób klas pokazało, że takie kombinacje nie są na tyle dobre, aby uznać je za poprawne. Dzieje się tak chociażby ze względu na to, że mapowanie z histogramu dało lepszy rezultat.

Dla pierwszego uruchomienia, przypisanie klas z 3.3 jest najlepszym mapowaniem. Na tych danych algorytm klasteryzacji został uruchomiony 5 razy, na trzech różnych progach ilości przeprowadzonych iteracji: 10, 100 i 400. Tak prezentują się wyniki poszczególnych uruchomień.

Tab. 3.4. Najlepsze wyniki dla 5 uruchomień na zbiorze danych "test"

Lp.	Najlepsza trafność dla 10 iteracji	Najlepsza trafność dla 100 iteracji	Najlepsza trafność dla 400 iteracji
1	53.38 % - Zamiana klas	50.63 % - Histogram	53.71 % - Histogram
2	62.38 % - Zamiana klas	58.92 % - Histogram	53.62 % - Histogram
3	33.33 % - Zamiana klas	52.86 % - Histogram	53.34 % - Zamiana klas
4	64.07 % - Zamiana klas	52.76 % - Zamiana klas	53.33 % - Histogram
5	37.75 % - Zamiana klas	59.68 % - Histogram	59.95 % - Histogram



Rys. 3.5. Uruchomienie 1: Macierze błędów po dokonaniu zamiany klas

Średnie procentowe wyniki trafności na wyznaczonych progach są do siebie dość podobne. Dla 10 iteracji średnia trafności jest mniejsza o 4 %. Można również zauważyć pewną zależność: czym mniej iteracji tym lepsza trafność prezentowana była w sposobie weryfikacji zamiany klas. Natomiast w liczbie iteracji od 100 do 400, sposób mapowania zwykle opierał się na bazie histogramu.

Zastosowanie sposobów weryfikacji na zbiorze danych zawierających 30000 dokumentów tekstowych dało gorsze wyniki przy uruchomieniu na 10 iteracjach.

Tab. 3.5. Najlepsze wyniki dla 3 uruchomień na zbiorze danych "train"

Lp.	Najlepsza trafność dla 10 iteracji	Najlepsza trafność dla 100 iteracji	Najlepsza trafność dla 400 iteracji
1	38.66 % - Zamiana klas	51.7 % - Histogram	34.53 % - Zamiana klas
2	33.45 % - Zamiana klas	34.54 % - Zamiana klas	51.77 % - Zamiana klas
3	43.72 % - Zamiana klas	51.8 % - Zamiana klas	51.66 % - Zamiana klas

Rozdział 4

Podsumowanie

W pracy zostały poruszone tematy związane z analizą tekstu oraz grupowaniem dokumentów tekstowych, wykorzystując w tym celu metody czyszczenia, normalizacji oraz reprezentacji tekstu za pomocą miary $TF - IDF$. Na dzień dzisiejszy, analizowanie dokumentów tekstowych jest dość trudnym zagadnieniem. Oprócz samej budowy zdań w tekście, należy również rozumieć sens słów, ponieważ w niektórych sytuacjach to właśnie sens słowa jest istotniejszym elementem, decydującym o przynależności dokumentu do konkretnej grupy tekstów.

Praca ta prezentuje poszczególne kroki podjęte dla osiągnięcia celu. Omówione zostały pojęcia teoretyczne konkretnych technologii, instalacja oraz konfiguracja środowiska pracy Visual Studio Code oraz języka programowania Python. Opisane zostały możliwości wykorzystywanych bibliotek do wstępnego przetwarzania tekstu.

W kolejnych częściach pracy przedstawione zostały etapy przygotowań dokumentów tekstowych do wykorzystania podczas grupowania tekstu, oraz algorytm, który został wykorzystany do osiągnięcia zamierzonego celu.

Aby dokonać weryfikacji dokładności klasteryzacji dokumentów tekstowych opracowano metodę dopasowywania klastrów wygenerowanych przez algorytm k-means do etykiet referencyjnych. Do tego celu wykorzystane zostało przedstawienie danych na histogramach, oraz zamiana klas wyjściowych z wykorzystaniem różnych mapowań.

Przedstawiony w niniejszej pracy system jest możliwy do uruchomienia na innych zbiorach dokumentów tekstowych. Przykładowe zbiory takich dokumentów dostępne są w załączniku na płycie CD dołączonej do pracy.

Bibliografia

[1] Charu C. Aggarwal. *Data Mining: The Textbook*. Springer, 2015.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[4] Jason Brownlee. How to clean text for machine learning with python. `https://machinelearningmastery.com/clean-text-machine-learning-python/`, 2017. [Online; accessed 01-February-2018].

[5] Wikipedia. Date format by country — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Date%20format%20by%20country&oldid=822208205`, 2018. [Online; accessed 01-February-2018].

[6] Rada Języka Polskiego. Skróty nazw miesięcy. `http://www.rjp.pan.pl/index.php?option=com_content&view=article&id=978:skroty-nazw-miesicy&catid=44&Itemid=145`, 2005. [Online; accessed 01-February-2018].

[7] Jana Vembunarayanan. Tf-idf and cosine similarity. `https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/`, 2013. [Online; accessed 02-February-2018].

[8] J MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symp. Math. Stat. Probab. Vol. 1 Stat.*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[9] Jörg Sander Xiaowei Xu Martin Ester, Hans-Peter Kriegel. A Density-Based Algorithm for Discovering Custers in Large Spatial Databases with Noise. pages 226–231, Institute for Computer Science, 1996. University of Munich.

[10] Xiang Zhang. Textclassificationdatasets. `https://drive.google.com/drive/u/0/folders/0Bz8a_Dbh9Qhbfll6bVpmNUtUcFdjYmF2SEpmZUZUcVNiMUw1TWN6RDV3a0JHT3kxLVhVR2M`, 2015. [Online; accessed 03-February-2018].