POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

# KIERUNEK: Informatyka

SPECJALNOŚĆ: Inżynieria Internetowa

## PRACA DYPLOMOWA

## INŻYNIERSKA

Klasyfikacja dokumentów tekstowych w środowisku Python

Classification of text documents in Python

AUTOR:

Piotr Chmiel

PROWADZĄCY PRACĘ:

dr. hab. inż. Henryk Maciejewski,

Zakład Systemów Komputerowych i Dyskretnych

OCENA PRACY:

### WROCŁAW 2015

Spis Treści

Spis Tabel 3

Spis Rysunków 4

Spis Listingów 5

1. Wprowadzenie 6

1.1. Cel i zakres pracy 6

2. Klasyfikacja dokumentów – analiza teoretyczna 8

2.1. Budowa modelu klasyfikatora 8

2.2. Wybór cech z dokumentów 9

2.3. Metody ograniczania rozmiaru wektorów cech 12

2.3.1 Znakownie części mowy 12

2.3.2 Analiza słowotwórcza – stemming i lematyzacja 13

3. Charakterystyka procesu przetwarzania tekstu i klasyfikacji doumentów – implementacja… 15

3.1. Pobieranie zbiorów dokumentów 16

3.2. Tworzenie korpusu artykułów 17

3.3. Generowanie i ograniczanie rozmiaru macierzy cech 21

3.4. Trenowanie modeli klasyfikatorów 24

4. Badania skuteczności klasyfikacji oraz rozmiaru macierzy cech 28

4.1. Miary skuteczności klasyfikacji 28

4.2. Wpływ zastosowania metod ograniczania wymiarowości na rozmiar macierzy cech 30

4.3. Porównanie skuteczności klasyfikatorów 32

4.4. Wpływ zmiany rozmiaru macierzy cech na skuteczność klasyfikacji 37

5. Aplikacja Kliencka 41

6. Podsumowanie 44

Bibliografia 28

**Spis Tabel**

[Tabela 1 Rozmiar zbiorów tekstów wyznaczonych z przestrzeni dokumentów 28](#_Toc435271016)

[Tabela 2 Wyniki prawdziwe i fałszywe kategorii sport 29](#_Toc435271017)

[Tabela 3 Zmiana rozmiaru macierzy cech 31](#_Toc435271018)

[Tabela 4 Porównanie trafności modeli klasyfikatorów 32](#_Toc435271019)

[Tabela 5 Porównanie precyzji klas klasyfikatorów 33](#_Toc435271020)

[Tabela 6 Porównanie wrażliwości klas modeli klasyfikatorów 33](#_Toc435271021)

[Tabela 7 Porównanie miary F klas modeli klasyfikatorów 34](#_Toc435271022)

[Tabela 8 Macierz strat SVM - reprezentacja macierzy binarna 36](#_Toc435271023)

[Tabela 9 Macierz strat SVM - reprezentacja macierzy zliczanie słów 36](#_Toc435271024)

[Tabela 10 Macierz strat SVM - reprezentacja macierzy optymalizacja TFIDF 36](#_Toc435271025)

[Tabela 11 Porównanie klasyfikatorów w zależności od zmiany rozmiaru binarnej macierzy cech 37](#_Toc435271026)

[Tabela 12 Wpływ ograniczenia wymiarowości binarnej macierzy cech na miarę f algorytmu Bernoulli Naive Bayes 39](#_Toc435271027)

**Spis Rysunków**

[Rysunek 1 Klasyfikacja nadzorowana - schemat procesu 8](#_Toc435271102)

[Rysunek 2 Binarna macierz cech 10](#_Toc435271103)

[Rysunek 3 Macierz zliczania słów 10](#_Toc435271104)

[Rysunek 4 Macierz zliczania słów optymalizowana metodą tfidf 11](#_Toc435271105)

[Rysunek 5 Stemming i lematyzacja 14](#_Toc435271106)

[Rysunek 6 Diagram komponentów systemu uczenia i testowania modeli klasyfikatorów 15](#_Toc435271107)

[Rysunek 7 Diagram maszyny stanowej - moduł "Web Crawler" 17](#_Toc435271108)

[Rysunek 8 Sposób działania funkcji word\_tokenize z biblioteki NLTK 19](#_Toc435271109)

[Rysunek 9 Diagram maszyny stanowej - tworzenie wektora cech pojedynczego dokumentu 23](#_Toc435271110)

[Rysunek 10 Diagram czynności - sposób działania aplikacji klienckiej 41](#_Toc435271111)

[Rysunek 11 Aplikacja kliencka - rezultat klasyfikacji 43](#_Toc435271112)

**Spis Listingów**

[Listing 1 Tworzenie korpusu artykułów 18](#_Toc435271164)

[Listing 2 Podział dokumentu na słowa 19](#_Toc435271165)

[Listing 3 Wyrażenia regularne wykorzystane w procesie normalizacji tekstu 20](#_Toc435271166)

[Listing 4 Tworzenie zbiorów trenujących i testowych 20](#_Toc435271167)

[Listing 5 Tworzenie macierzy cech 21](#_Toc435271168)

[Listing 6 Wybrane metody klasy FeatureExtractorPos 22](#_Toc435271169)

[Listing 7 Wyszukiwanie kolokacji 24](#_Toc435271170)

[Listing 8 Fragment kodu źródłowego klasy SklearnClassifier 24](#_Toc435271171)

[Listing 9 Trenowanie modeli klasyfikatorów 25](#_Toc435271172)

[Listing 10 Parametry wybranych klasyfikatorów podczas procesu uczenia 26](#_Toc435271173)

[Listing 11 Klasyfikacja tekstu - aplikacja kliencka 42](#_Toc435271174)

1. **Wprowadzenie**

W sierpniu 2014 roku firma Netcraft ogłosiła przekroczenie liczby jednego biliona stron w Internecie[[1]](#footnote-1). Codziennie powstają miliony cyfrowych dokumentów, a istniejące papierowe ulegają procesowi digitalizacji. W związku z ogromną liczbą tekstów przetwarzanych przez człowieka obserwuje się wzrost zapotrzebowania ich uporządkowania i klasyfikacji do różnych kategorii tematycznych. Grupowanie dokumentów ułatwia ich późniejszą analizę i przyspiesza wyszukiwanie. Wyznaczenie właściwej etykiety może być wykonane przez ludzi, ale w przypadku bardzo dużej liczby danych byłoby bardzo kosztowne i czasochłonne. Można sobie wyobrazić duże przedsiębiorstwo programistyczne, do którego codziennie przychodzi kilkaset wiadomości do skrzynki pocztowej działu pomocy technicznej. Następnie pracownik obsługujący i czytający e-maile przekazuje je do właściwych osób zajmujących się problemem wynikającym z treści listu. Efektywność zakładu pracy wzrosłaby, gdyby proces analizy treści wykonywał się automatycznie. Wówczas firma mogłaby wykorzystać pracownika do innych celów. Niemal każdy użytkownik poczty elektronicznej spotyka się z niechcianymi wiadomościami tzw. SPAM. Zbudowanie wydajnego filtra, także należy do zagadnień klasyfikacji. Codzienne problemy sprawiły, że popularność zdobywa dziedzina uczenia maszyn, która może zostać wykorzystana do zbudowania modelu klasyfikatora automatyzującego pracę człowieka. Wówczas jego rola sprowadzi się wyłącznie do przypisania klas przykładom trenującym.

* 1. **Cel i zakres pracy**

Głównym celem pracy było wykonanie oprogramowania umożliwiającego nauczenie i przetestowanie klasyfikatorów dokumentów tekstowych w języku angielskim. System został zrealizowany w oparciu o biblioteki „Natural Language Toolkit” oraz „Scikit-Learn” języka Python realizujących wybrane zdania przetwarzania języka naturalnego. Korzystając z tych narzędzi opracowano program zapewniający wygenerowanie wektorów cech na podstawie dokumentów, redukcję jego wymiarowości oraz stworzenie modeli wybranych klasyfikatorów. Powstałe klasyfikatory przewidują klasy artykułów informacyjnych i popularnonaukowych ze zbioru kategorii Sport, Polityka i Zdrowie. Zakres pracy:

* zapoznanie się z narzędziami do przetwarzania języka naturalnego w języku Python,
* zgromadzenie danych uczących i testowych ( dokumenty z wymienionych wcześniej grup tematycznych pochodzących z brytyjskich i amerykańskich internetowych portali informacyjnych ),
* opracowanie skryptów generujących cechy z dokumentów,
* implementacja wybranych metod redukcji wymiarowości wektorów cech,
* nauczenie wybranych klasyfikatorów,
* przeprowadzenie testów skuteczności klasyfikacji.

Dodatkowo w ramach pracy powstał interfejs webowy umożliwiający wykorzystanie powstałych modeli klasyfikatorów do przewidywania klas dokumentów wprowadzonych przez użytkownika.

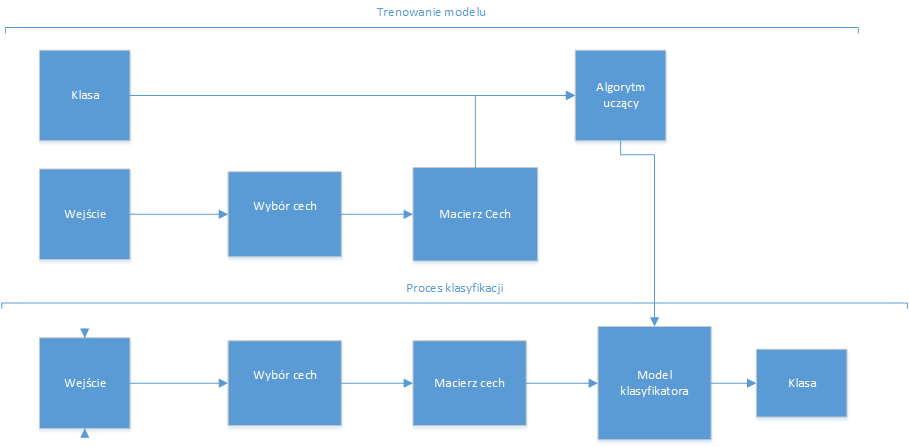
1. **Klasyfikacja dokumentów – analiza teoretyczna**

Klasyfikacją tekstu określa się przyporządkowanie dokumentu , gdzie jest przestrzenią dokumentów do zróżnicowanego zbioru klas . Klasy nazywane są także etykietami oraz kategoriami. W podstawowych zadaniach klasyfikacji każdy dokument z przestrzeni rozważany jest osobno, a zbiór klas definiuje się z góry. W najprostszym przypadku etykiety są rozłączne. Dziedzina klasyfikacji występuje w wielu interesujących wariantach. Klasyfikacja binarna opiera się na decyzji pomiędzy dwoma kategoriami z przestrzeni takich jak prawda lub fałsz. Często odpowiada na pytania np. czy dana wiadomość należy do niechcianej poczty. Klasyfikacja wieloklasowa, która zostanie omówiona szerzej w niniejszej pracy polega na dopasowaniu danych wejściowych do jednej ze skończonej liczby co najmniej dwóch klas. Klasyfikacja wielowyjściowa przyporządkowuje dokumenty do więcej niż jednej klasy. Wyróżniamy także klasyfikację sekwencyjną, gdzie dane przetwarzane są wspólnie oraz klasyfikację „otwartych klas”, w której kategorie nie są zdefiniowane z góry.

* 1. **Budowa modelu klasyfikatora**

Jednym ze sposobów stworzenia modelu klasyfikatora jest uczenie nadzorowane ( ang. supervised learning ). W tej metodzie człowiek nazywany nauczycielem lub dozorcą wykorzystuje algorytm uczący do trenowania klasyfikatora lub funkcji klasyfikującej , która zamienia dokumenty wyjściowe w klasy:

Wiele znanych algorytmów uczących (np. Regresja Logistyczna) wykorzystuje funkcję . Rysunek 1 przedstawia proces klasyfikacji nadzorowanej.



Rysunek Klasyfikacja nadzorowana - schemat procesu[[2]](#footnote-3)

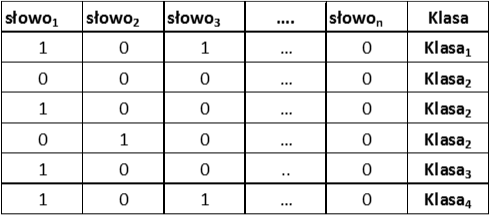
W pierwszym kroku nauczyciel decyduje się na wybór zbioru trenującego. W przypadku klasyfikacji dokumentów tekstowych powinny być to teksty reprezentujące każdą klasę ze zbioru . Należy wziąć pod uwagę dwa aspekty wpływające na wydajność klasyfikatora trenowanego skończoną liczbą próbek n: odchylenie i wariancję. Odchyleniem nazywamy błąd wynikający z założeń algorytmu uczącego. Wysokie odchylenie może spowodować, że metoda ucząca nie znajdzie odpowiednich relacji pomiędzy cechami i dozorca zetknie się z problemem niedouczenia ( ang. underfitting ). Wariancja jest błędem związanym z wrażliwością na różnorodność w zbiorze trenującym. Duża wariancja może powodować przeuczenie ( ang. overfitting ) klasyfikatora. Wówczas model przewiduje prawidłowe klasy w zbiorze trenującym, natomiast źle w tekstach pochodzących spoza tego zbioru. Jednym z czynników wpływających na odchylenie i wariancję jest rozmiar zbioru trenującego. Jeżeli funkcja trenująca algorytmu posiada prostą formę, wtedy klasyfikator będzie mógł nauczyć się z małej liczby dokumentów o dużym odchyleniu i małej wariancji. W przypadku złożonej funkcji trenującej najlepiej sprawdzi się wysoka liczba tekstów o małej wariancji i dużym odchyleniu. Zaawansowane algorytmy uczące potrafią automatycznie dostosowywać kompromis pomiędzy wariancją o odchyleniem danych wejściowych i w zależności od ich liczby dobierać odpowiednią złożoność funkcji . Nie ma określonych jednoznacznie zasad dotyczących rozmiaru zbioru uczącego. Jego wielkość powinna być eksperymentalnie dostosowana do algorytmu uczącego. Należy pamiętać, aby zbiór był reprezentatywny tzn. liczba przedstawicieli każdej klasy powinna być porównywalna.

Kolejny krok dotyczy wyboru cech dokumentów. Wykorzystując przetwarzanie języka naturalnego dozorca tworzy wektory cech oznaczone klasą, z której pochodzi dokument. Tablice cech łączone są w macierz, której kolumnami są cechy, a wierszami etykiety dokumentów. Macierz cech podawana jest jako wejście metody uczącej. Proces przetwarzania przez algorytm macierzy nazwano trenowaniem. W jego wyniku powstaje model klasyfikatora zdolny do przewidywania etykiet dowolnych dokumentów wejściowych.

Zbudowany model może zostać wykorzystany do klasyfikacji dowolnego dokumentu. Należy pamiętać, że klasyfikator przyjmuje macierz, dlatego dane należy uprzednio przetworzyć podobnie jak w procesie trenowania. W przypadku większości algorytmów uczących powstały model, gdy na wejściu otrzyma dokument nie przynależący do zbioru klas tego klasyfikatora przypisze tekst do jednej z kategorii.

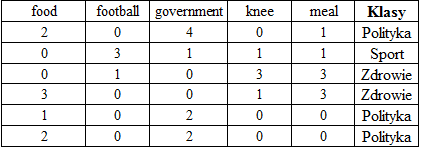
* 1. **Wybór cech z dokumentów**

Ważnym elementem w procesie analizy przestrzeni zebranych dokumentów jest dziedzina przetwarzania języka naturalnego. Jako język naturalny określa się język do codziennej komunikacji pomiędzy ludźmi np. Polski lub Angielski. Podczas procesu uczenia modelu klasyfikatora należy przetworzyć dokumenty i wybrać z nich cechy. Cechami w tekstach są słowa. Najpopularniejszym sposobem wyboru cech jest tzw. „bag of words”. Metoda polega na wyłuskaniu wszystkich słów ze zbioru tekstów i umieszczeniu ich w macierzy dokumentów. Każdy wiersz macierzy reprezentuje wektor cech dokumentu ze zbioru wybranego z przestrzeni . Kolumny wyrażają wyrazy (cechy), wybrane wcześniej z całego zbioru tekstów. Ostatnia kolumna przedstawia klasę dokumentu danego wiersza. Komórki macierzy mogą być reprezentowane na wiele sposobów. Pierwszym z nich jest binarna tabela cech pokazana na rysunku 2.

****

Rysunek Binarna macierz cech

Jeżeli słowo i występuje dokumencie j, komórka [i,j] zostaje wypełniona wartością logiczną 1, w przeciwnym wypadku 0. Kolejnym sposobem uzupełnienia pól macierzy jest zliczanie cech przedstawione na rysunku 3. Wówczas w komórce [i,j] znajduje się liczba wystąpień słowa i w tekście j.

****

Rysunek Macierz zliczania słów

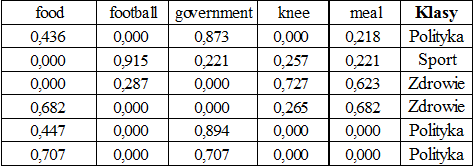
Sposób zliczania słów, można optymalizować stosując algorytm tfidf Metoda należy do dziedziny statystki numerycznej. Odpowiada na pytanie o wagę słowa w stosunku do dokumentu w zbiorze. Oprócz klasyfikacji posiada zastosowanie w ocenianiu trafności podczas realizacji zapytań w wyszukiwarkach internetowych Wyrażana jest wzorem:

.

Tf (ang. term frequency) oznacza częstość termów opisaną wzorem:

gdzie informuje o liczbie wystąpień cechy i w dokumencie j, a mianownik reprezentuje sumę wystąpień wszystkich k słów w rozważanym tekście. Idf ( ang. inverse document frequency ) określa odwrotną częstość dokumentów, wyrażaną wzorem:

gdzie D oznacza zbiór dokumentów w macierzy, natomiast liczbę dokumentów zawierających co najmniej 1 wystąpienie cechy . Rysunek 4 przedstawia macierz z rysunku 3 po zastosowaniu algorytmu tfidf.



Rysunek Macierz zliczania słów optymalizowana metodą tfidf

Z punktu widzenia wyboru cech warto wspomnieć o kolokacjach, sekwencjach słów będących częstym zestawieniem, niosących ze sobą specjalne znaczenie. Przykładem takiego związku jest „New York”, nazwa miasta w Stanach Zjednoczonych. Podczas tworzenia macierzy oprócz pojedynczych słów jako cechy wyodrębnia się kolokacje. Jako najbardziej istotną w przetwarzaniu wyrazów występujących często obok siebie uznaje się miarę asocjacji ( ang. measure of association ), która ocenia czy słowa współwystępują wyłącznie przez przypadek czy ma to znaczenie statystyczne. Powszechnie stosowane miary to  mutual information, t scores, i log-likelihood.

* 1. **Metody ograniczania rozmiaru wektorów cech**

W rozdziale 2.1 zostały wymienione miary odchylenia i wariancji mające wpływ na skuteczność trenowania algorytmu uczącego. Na powyższe metryki oddziałuje, także rozmiar macierzy cech. Duża liczba cech zwiększa wariancję i tym samym może spowodować przeuczenie klasyfikatora, a mała powoduje wzrost odchylenia, co prawdopodobnie doprowadzi do niedouczenia modelu. Co więcej w macierzy wykorzystującej metodę bag of words, kolumny reprezentują wszystkie słowa ze zbioru dokumentów. Jednak część z nich z punktu widzenia klasyfikacji nie niesie ze sobą żadnych informacji. Są to np. traktowane w oderwaniu od kontekstu liczebniki, zaimki lub przymiotniki. Zbyt duży rozmiar macierzy powoduje wysokie obciążenie pamięci komputera. W celu poprawy skuteczności klasyfikacji i zmniejszenia zajmowanych zasobów stosuje się metody ograniczania wymiarowości wektorów cech podczas tworzenia macierzy.

* + 1. **Znakowanie części mowy**

Słowa można podzielić na klasy. Klasy w tym przypadku nazywane są częściami mowy. W odniesieniu do ograniczania wymiarowości wektorów cech jednym ze sposobów wykluczania słów jest wybór kategorii gramatycznych, które z punktu widzenia nauczyciela nie niosą ze sobą pożądanej informacji, następnie oznakowanie częściami mowy wszystkich słów, a w efekcie usunięcie tych nie przynależących do klas wybranych w kroku 1. Do tego celu należy wykorzystać algorytmy znajdowania odpowiednich klas gramatycznych. Gramatyka każdego języka różni się od siebie, dlatego przytoczone niżej przykłady będą dotyczyć języka angielskiego.

Najprostszy możliwy moduł znakujący części mowy przypisuje taką samą klasę każdemu ze słów. Nawet w przypadku wyboru najczęściej występującej kategorii gramatycznej jaką w języku angielskim jest rzeczownik, obserwuje się bardzo niską wydajność tego rozwiązania. Jednak pomimo swojej prostoty stanowi ono podstawę pozostałych algorytmów. Inne podejście oparto na wyrażeniach regularnych. W tej metodzie nauczyciel tworzy listę reguł np. „wszystkie słowa kończące się na „ing” są czasownikami”. Znakowanie odbywa się według stworzonych zasad. W przypadku języka angielskiego ten sposób znakowania nie sprawdza się, ponieważ wiele wyrazów w zależności od kontekstu użycia w zdaniu lub wypowiedzi zmienia własne znaczenie oraz kategorię gramatyczną. Co więcej słowa nie spełniające żadnej z reguł nie zostaną oznakowane. Kolejny algorytm polega na zbudowaniu modelu n najczęściej występujących wyrazów w danym języku oraz oznakowaniu ich przez człowieka. Dobór etykiety odbywa się na podstawie zbudowanego zbioru. Jeżeli słowo nie występuje w modelu nie zostanie oznakowane. Reprezentacją modelu może być, także treść książki podzielona na części mowy przez nauczyciela. Wówczas, podczas znakowania moduł zawsze wybierze część mowy, która dla danego słowa występowała najczęściej w modelu.

Najbardziej zaawansowaną techniką znakowania kategorii gramatycznej jest tzw. n-gram tagging, czyli wybór części mowy na podstawie słów z otoczenia. Za kontekst słowa przyjmuje się wówczas n-1 poprzedzających wyrazów. Podobnie jak w przypadku wyżej wymienionych metod należy stworzyć model oparty o wcześniej oznakowany tekst. Proces przekazywania modelu do modułu znakującego określa się trenowaniem. Rezultatem znakowania etykiety danego słowa znajdującego się w sąsiedztwie n-1 posiadających przypisaną klasę będzie najczęściej występująca część mowy tego samego wyrazu znajdująca się w dokładnie tak samo oznakowanym otoczeniu w przyjętym modelu.[[3]](#footnote-4)

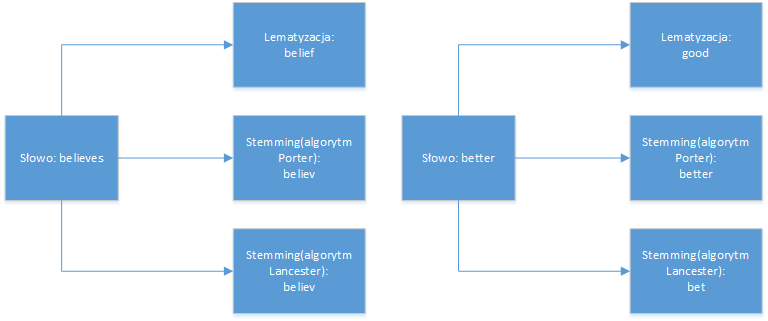
Większość przedstawionych powyżej rozwiązań posiada jedną wadę: istnieje możliwość, że metoda znakująca nie będzie w stanie określić części mowy. Z tego powodu stosuje się rozwiązania oparte na połączeniu kilku algorytmów. Nauczyciel umieszcza metody na liście priorytetów, którą kończy podstawowa przypisująca słowom zawsze najczęściej występującą część mowy w danym języku. Moduł znakujący rozpoczyna od początku listy. Jeżeli wybrany algorytm nie jest w stanie określić kategorii gramatycznej, wykorzystuje się kolejny, aż do wyczerpania zestawienia. Tym sposobem każdemu słowu zostanie przypisana część mowy, ponieważ ostatni sposób zawsze zwróci kategorię ( w przypadku rozważanego języka angielskiego będzie to rzeczownik ).

* + 1. **Analiza słowotwórcza – stemming i lematyzacja**

Z przyczyn gramatycznych w języku angielskim wyrazy występują w różnych formach. Czasowniki ze wzglądu na czas, w którym występują różnią się od siebie przyrostkiem np. abandon, abandoned, abandoning. Rzeczowniki mogą przybrać formę liczby pojedynczej i mnogiej np. cat i cats. Uogólniając, każdy wyraz składa się podstawy słowotwórczej oraz opcjonalnie dodawanego formantu zwanego, także zrostem lub afiksem. Afiksy podzielono na fleksyjne i derywacyjne. Derywacyjne zmieniają część mowy wyrazu np. differ i different. Fleksyjne zmieniają formę, ale nie zmieniają kategorii gramatycznej.

Proces oddzielania podstawy słowotwórczej od zrostu nazywa się stemmingiem. Najpopularniejszymi algorytmami stosowanymi w języku angielskim w celach wyodrębnienia rdzenia słowa są Porter stworzony przez Martina Portera i Lancester rozwijany przez Uniwersytet w Lancester. Nie udowodniono wyższości jednego algorytmu nad drugim.

Operacją podobną do stemmingu jest lematyzacja. Polega na znajdowaniu rdzenia słowa. Rdzeniem w tym przypadku nie określa się podstawy słowotwórczej, a wyraz od którego słowo pochodzi (ang. origin ). W przypadku zastosowania stemmingu zawsze otrzymujemy wyraz o tym samym znaczeniu, natomiast w przypadku lematyzacji może się ono zmienić. Obie metody wykorzystuje się w wyszukiwarkach internetowych w celu zbudowania wydajnego indeksu słów. Różnice pomiędzy tymi dwoma operacjami przedstawia rysunek 5.

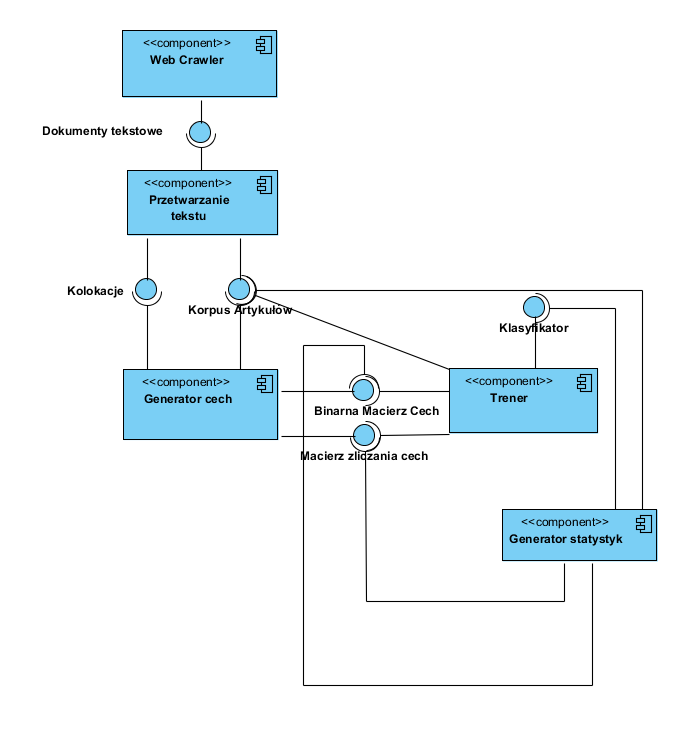


Rysunek Stemming i lematyzacja

Z punktu widzenia klasyfikacji tekstu biorąc pod uwagę słowa z tej samej grupy różniące się formantem lub będące kolejnym stopniem tego samego przymiotnika, ich rdzenie niosą tą samą informację. W procesie ograniczania wymiarowości macierzy cech, do grupowania i składania tych samych słów w jedną cechę stosuje się kombinację operacji stemmingu i lematyzacji.

1. **Charakterystyka procesu przetwarzania tekstu i klasyfikacji dokumentów – implementacja**

Niniejszy rozdział zawiera opis realizacji projektu, którego założenia zostały przedstawione w rozdziale 1.1. Powstał system umożliwiający budowę modeli klasyfikatorów oraz testowania ich skuteczności. Jego komponenty zostały napisane w języku Python 3 w oparciu o bibliotekę NLTK w wersji 3.0.4 oraz Sklearn w wersji 0.16.1. „Natural Language Toolkit” wykorzystano do przetwarzania tekstu, natomiast „Scikit-Learn” dostarczyło algorytmy klasyfikacji, które potem wzięły udział w procesie trenowania. Na rysunku 6 został pokazany diagram komponentów. Przedstawia wszystkie elementy architektury zaimplementowanego rozwiązania.

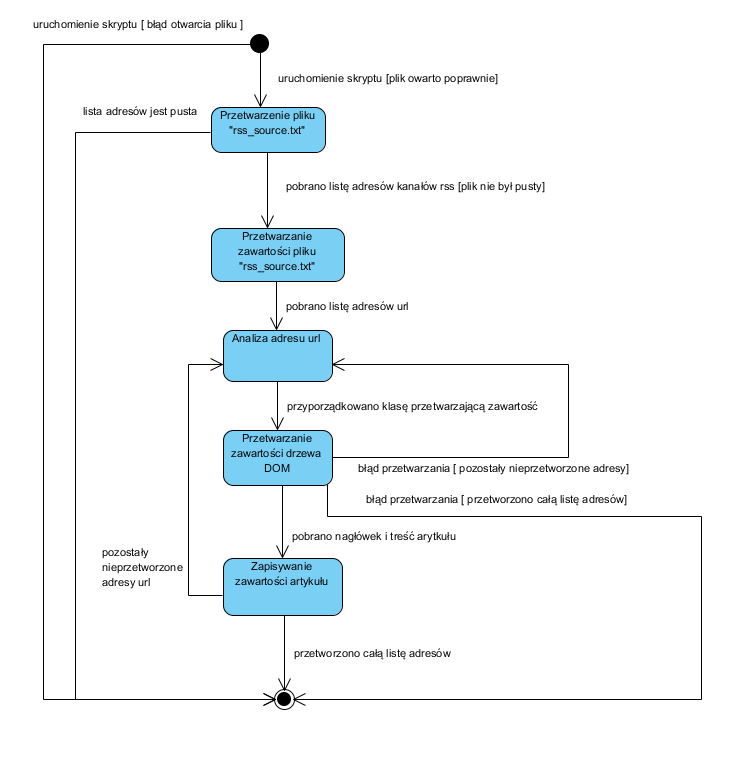


Rysunek Diagram komponentów systemu uczenia i testowania modeli klasyfikatorów

Pierwszy z nich „Web Crawler” jest reprezentacją maszyny automatycznie pobierającej artykuły w języku angielskim z popularnych portali internetowych. Moduł „Przetwarzanie tekstu” generuje oznaczony klasami korpus dokumentów, udostępnia funkcje umożliwiające znajdowanie kolokacji, stworzenie zbiorów trenujących i testowych, a także podział tekstów na słowa. „Generator cech” tworzy oraz ogranicza rozmiar macierzy cech stosując techniki opisane w rozdziale 2. „Trener” służy do stworzenia modelu klasyfikatora, który zostaje poddany serializacji za pomocą modułu pickle z biblioteki standardowej języka Python, a następnie zapisany do pliku. Takie podejście umożliwia wykorzystanie klasyfikatora w dowolnym innym systemie lub aplikacji. „Generator statystyk” umożliwia przeprowadzenie testów skuteczności klasyfikacji.

* 1. **Pobieranie zbiorów dokumentów**

W podanych wcześniej założeniach ustalono, że dokumenty będą pobierane z trzech kategorii tematycznych: Sport, Polityka i Zdrowie. Do tego celu zostały wykorzystane kanały rss znanych anglojęzycznych i amerykańskich stron internetowych. Przetwarzanie wiadomości rss umożliwiła biblioteka feedparser w wersji 5.2.1. Użytkownik umieszcza listę kanałów w pliku „rss\_source.txt”. Każdy kanał musi znajdować się w nowej linii oraz zostać poprzedzony nazwą klasy tematycznej pisaną wielkimi literami. Jeżeli linia rozpoczyna się od znaku „#” znajdujący się w niej adres zostaje pomijany w przetwarzaniu. Preferowane adresy kanałów powinny pochodzić z organizacji „Yahoo”, „BBC”, „Reuters”, „Telegraph”, „Fox News”, „Webmd” i „Daily Mail”. Adresy innych portali, także zostaną przetworzone, jednak artykuły mogą z nich zostać pobrane nieprawidłowo. Moduł „Web Crawler” odczytuje kolejno zawartość kanałów rss z listy. W rezultacie otrzymywane są pliki xml. Z nich program pobiera adresy url stron, na których znajdują się teksty. Następnie następuje analiza każdego z nich. Na podstawie adresu url artykułu dobierana jest odpowiednia klasa przetwarzająca drzewo DOM pod kątem pobrania tytułu i treści wiadomości. Jeżeli tekst został pobrany poprawnie zostaje zapisany do folderu z nazwą klasy tematycznej, do której przynależy. Nazwą pliku jest temat artykułu. W przypadku gdy w wyżej wymienionym katalogu znajduje się już plik o tej samej nazwie, nowy tekst zostaje pomijany. Domyślnie foldery kategorii tworzone są w katalogu „Articles”. Na rysunku 7 został przedstawiony diagram ilustrujący działanie modułu „Web Crawler”. Szczególną uwagę warto zwrócić na akcję powodującą przejście ze stanu „Analiza adresu url” do „Przetwarzanie zawartości drzewa DOM”. Następuje wówczas przyporządkowanie klas przetwarzających źródło strony internetowej. Klasa dobierana jest na



Rysunek Diagram maszyny stanowej - moduł "Web Crawler"

podstawie zawartości słowa kluczowego ( nazwy organizacji ) w adresie url. Klasy zostały dostosowane indywidualnie do każdej ze stron preferowanych firm. Jeżeli nie zostanie znaleziona dedykowana klasa, zostanie przyporządkowana domyślna. W jej przypadku za treść artykułu uznaje się tekst znajdujący wewnątrz wszystkich znaczników p wybranego pliku html. W związku z tym mogą zostać pobrane nadmiarowe informacje nie należące do treści wiadomości. Wówczas użytkownik ponosi odpowiedzialność za walidację danych wyjściowych modułu.

* 1. **Tworzenie korpusu artykułów**

Jednym z podstawowych składników biblioteki Natural Language Toolkit jest korpus – uporządkowany zbiór tekstów. Programista może skorzystać z dostępnych kolekcji dokumentów lub stworzyć własną na podstawie dowolnych danych tekstowych. Skrypt „Web Crawler” pobiera dokumenty do podfolderów katalogu „Articles” oznaczonych klasą tematyczną tekstu. Dostęp do dowolnego tekstu wymaga za każdym razem podania ścieżki do dokumentu oraz wyłuskania nazwy katalogu, w którym się znajduje. W celu ułatwienia przetwarzania i zarządzania zbiorem został stworzony dedykowany korpus artykułów. Do tego celu została wykorzystana klasa CategorizedPlaintextCorpusReader z pakietu nltk.corpus.reader. Tworzenie korpusu przedstawia listing 1.

Listing Tworzenie korpusu artykułów

|  |
| --- |
| **from** nltk.corpus.reader **import** CategorizedPlaintextCorpusReader  training = CategorizedPlaintextCorpusReader(**"Articles"**, **r'.\*\.txt'**, cat\_pattern=**r'(\w+)'**, encoding=**"utf-8"**)   **def** print\_corpus\_info():  **print**(**"Training Corpus INFO"**)   **for** category **in** training.categories():  **print**(**"Number of documents in {0:8} category: {1}"**.format(category, len(training.fileids(category))))   **print**(**"\n"**) |

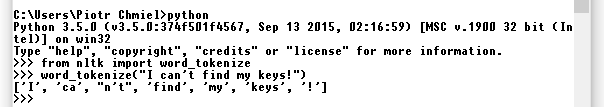
Pierwsze dwa argumenty konstruktora klasy zbioru oznaczają katalog podstawowy oraz pliki, które przekazywane są do funkcji tworzącej klasy nadrzędnej PlainTextCorpusRaeader wykonującego odczyt tekstu. Słowo kluczowe cat\_pattern trzeciego argumentu jest wyrażeniem regularnym wyłuskującym, ze ścieżki pliku klasę argumentu. Ostatni parametr mówi o trybie kodowania plików. W rezultacie powstaje obiekt zbioru tekstów. Klasa CategorizedPlaintextCorpusReader posiada szereg ciekawych możliwości. Za pomocą metody categories programista otrzymuje listę kategorii. Funkcja składowa fileids zwraca listę dokumentów, a jako opcjonalny parametr przyjmuje kategorię. Programista może wyświetlić listę wszystkich plików w zależności od etykiety dokumentu. Metoda open przyjmująca nazwę pliku z listy zwracanej przez fileids zwraca obiekt nltk.data.SeekableUnicodeStreamReader umożliwiający w prosty sposób dostęp do treści dowolnego dokumentu korpusu. Stworzenie korpusu umożliwiło efektywne zarządzanie pobranym zbiorem elementów.

Korpus zgromadzonych artykułów został częścią komponentu „Przetwarzanie tekstu”. Do modułu dołączono funkcje umożliwiające wykorzystanie zbioru tekstów pod kątem dalszego przetwarzania w kierunku budowy modeli klasyfikatora. Na listingu 2 została przedstawiona funkcja get\_words\_and\_replace przyjmująca nazwę dokumentu znajdującego się w korpusie i zwracająca listę słów podanego artykułu.

Listing Podział dokumentu na słowa

|  |
| --- |
| **from** nltk **import** word\_tokenize  **from** text\_processing.replacers **import** RegexpReplacer  **def** get\_words\_and\_replace(fileid):   content = training.open(fileid).read()  replacer = RegexpReplacer()    **return** list(word\_tokenize(replacer.replace(content))) |

Do podziału tekstu na słowa wykorzystano funkcję word\_tokenize z pakietu NLTK. Natural Language Toolkit zawiera kilka klas służących do wyłuskiwania wszystkich słów m. in. PunktWordTokenizer, WordPunktTokenizer, WhitespaceTokenizer, TreebankWordTokenizer i TwitterTokenizer. Podane klasy różnią się metodami dzielenia dokumentów na wyrazy, sposobem radzenia sobie z interpunkcją i kontrakcjami oraz wrażliwością na charakterystyczne elementy tekstów. Funkcja word\_tokenize stanowi opakowanie metody tokenize klasy TreebankWordTokenize. Polega na oddzielaniu słów za pomocą spacji i interpunkcji. Na rysunku 8 został przedstawiony sposób działania wykorzystanej funkcji podziału w interpreterze języka Python.



Rysunek Sposób działania funkcji word\_tokenize z biblioteki NLTK

Znaki interpunkcyjne nie zostały usunięte. Pozwala to klientowi programiście zdecydować jak należy z nimi postąpić. TreebankWordTokenizer podczas dzielenia dokumentów wykorzystuje konwencje przyjęte w korpusie PennTreebank. Kolekcja została stworzona w latach 80. XX wieku na podstawie artykułów z „Wall Street Journal”. Jedną z najbardziej znaczących konwencji jest sposób przetwarzania kontrakcji widoczny w przypadku „can’t” na rysunku 8. Taki sposób dzielenia uznano za niedopuszczalny. Z tego powodu w module „Przetwarzanie Tekstu” napisano klasę RegexpReplacer dokonującą normalizację tekstu przed przystąpieniem do podziału na słowa. Do tego celu wykorzystano wyrażenia regularne przedstawione na listingu 3. Kontrakcje zamieniane są według podanych zasad.

Listing Wyrażenia regularne wykorzystane w procesie normalizacji tekstu

|  |
| --- |
| replacement\_patterns = {  (**r'won\'t'**, **'will not'**),  (**r'can\'t'**, **'cannot'**),  (**r'i\'m'**, **'i am'**),  (**r'ain\t'**, **'is not'**),  (**r'(\w+)\'ll'**, **'g<1> will'**),  (**r'(\w+)n\'t'**, **'g<1> not'**),  (**r'(\w+)\'s'**, **'g<1> is'**),  (**r'(\w+)\'re'**, **'g<1> are'**),  (**r'(\w+)\'d'**, **'g<1> would'**), } |

Kolejną istotną funkcją znajdującą w module „Przetwarzanie tekstu” jest get\_training\_documents ( listing 4 ) zwracająca zbiór trenujący i testowy w postaci listy dwuelementowych krotek, gdzie pierwszy element reprezentuje lista słów artykułu, a drugi jego kategoria tematyczna. Funkcja przyjmuje dwa parametry: cut\_off informujący jaka część każdej klasy znajdzie się w zbiorze uczącym oraz save typu boolean mówiąca czy zbiory mają być zapisane do pliku w celu użycia w innych modułach po zakończeniu programu.

Listing Tworzenie zbiorów trenujących i testowych

|  |
| --- |
| **def** get\_training\_documents(cut\_off=0.75, save=False):  train\_set = []  test\_set = []   **for** category **in** training.categories():  category\_set = [(fileid, category) **for** fileid **in** training.fileids(category)]  shuffle(category\_set)  cut\_set = int(len(category\_set) \* cut\_off)  train\_set.extend(category\_set[:cut\_set])  test\_set.extend(category\_set[cut\_set:])   **if** save:  save\_documents(train\_set, **"train\_documents"**)  save\_documents(test\_set, **"test\_documents"**)   train\_set = [(get\_words\_and\_replace(fileid), category) **for** fileid, category **in** train\_set]  test\_set = [(get\_words\_and\_replace(fileid), category) **for** fileid, category **in** test\_set]   **if** save:  save\_documents(train\_set, **"train\_feature\_set"**)  save\_documents(test\_set, **"test\_feature\_set"**) |

Aby zapewnić losowość zawartości zbiorów wykorzystano funkcję shuffle z pakietu random biblioteki standardowej języka Python zmieniającą kolejność dokumentów w liście pobranej z korpusu artykułów.

* 1. **Generowanie i ograniczanie rozmiaru macierzy cech**

Proces tworzenia oraz ograniczania rozmiaru macierzy cech zrealizowano w komponencie „Generator Cech”. W tym module macierz reprezentuje lista dwuelementowych krotek, w których pierwszym elementem jest słownik z biblioteki standardowej języka Python, a drugim kategoria tematyczna artykułu. Słownik stanowi odpowiednik wektora cech dokumentu. Jego kluczami są słowa, natomiast ich wartości zależą od przyjętego sposobu reprezentacji macierzy. Macierz binarną opisują wyrazy oznaczone jedynką logiczną, a macierz zliczania słów te wyrażone liczbą wystąpień danego słowa w rozważanym tekście. Na listingu 5 został przedstawiony wycinek metody tworzenia zbiorów cech. Funkcja get\_training\_documents zwraca przypadki trenujące i testowe wstępnie podzielone na słowa. Są one podawane jako drugi argument metody apply\_features pakietu NLTK, która dla każdego elementu w zbiorze aplikuje funkcję tworzącą oznaczone wektory cech. Jej rezultat jest wykorzystywany do trenowania modelu klasyfikatora. Została stworzona klasa FeatureExtractorPos zajmująca się przetwarzaniem słów dokumentu i ograniczaniem rozmiaru wektora cech. Funkcje binary\_bag\_of\_word i counted\_bag\_of\_words stanowią opakowanie metody extract\_features wyżej wymienionej klasy zwracającej słownik cech. Zgodnie z przyjętymi nazwami każda z nich określa typ reprezentacji macierzy cech. W procesie zmniejszania wymiarowości wektorów wykorzystano znakowanie części mowy. Do tego celu zastosowano POS\_TAGGER z pakietu NLTK. Pod tą zmienną znajduje się obiekt modułu znakującego uznanego za najbardziej skuteczny przez twórców pakietu. Jako model wykorzystuje „Brown Corpus”, a podczas działania połączone metody opisane w rozdziale 2.3.

Listing Tworzenie macierzy cech

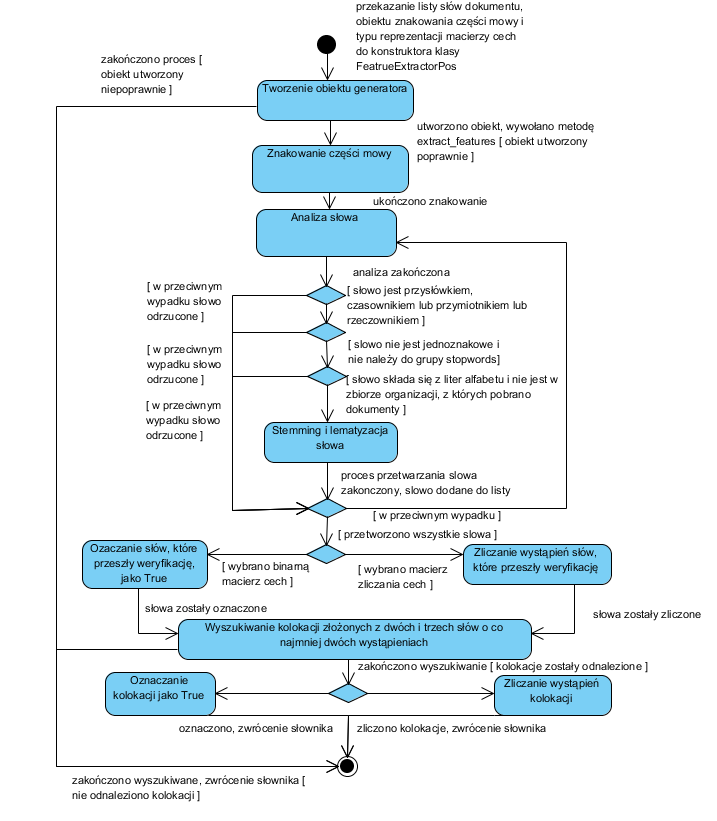
|  |
| --- |
| **from** nltk.classify **import** apply\_features  tagger = nltk.data.load(nltk.tag.\_POS\_TAGGER)  **def** binary\_bag\_of\_words(words, extractor=FeatrueExtractorPos, tagger=tagger):  extractor = extractor(words, tagger, binary=True)  **return** extractor.extract\_features()   **def** counted\_bag\_of\_words(words, extractor=FeatrueExtractorPos, tagger=tagger):  extractor = extractor(words, tagger, binary=False)  **return** extractor.extract\_features()  train\_documents, test\_documents = get\_training\_documents(cut\_off=0.75, save=True)  train\_set = apply\_features(binary\_bag\_of\_words, train\_documents) |

Przyjęto, że ze zbioru cech odrzucane będą słowa, które nie są czasownikami lub przymiotnikami lub przysłówkami lub rzeczownikami. Co więcej nie mogą one składać się z jednego znaku ani należeć do grupy stopwords pakietu NLTK. Do tego zbioru należą funktory np. „is”, „a”, „at”, „which” oraz frazy „There is”, „The Who”, które z punktu widzenia przetwarzania tekstu nie niosą żadnej informacji. Dodatkowo pomijane są wyrazy zawierające znaki nie należące do alfabetu oraz nazwy organizacji i firm, z których stron internetowych były pobierane artykułu. Realizację wyżej wymienionych zasad przedstawia generator języka Python „pos\_generator” należący do klasy FeatureExtractorPos znajdujący się na listingu 6. Słowo spełniające kryteria poddawano stemmingowi i lematyzacji z wykorzystaniem funkcji biblioteki Natural Language Toolkit. Do stemmingu wybrano algorytm Porter oraz wykorzystano część mowy rozważanego wyrazu, aby zwiększyć jakość procesu. W efekcie ograniczania liczby cech zwracane jest słowo pisane małymi literami. Powoduje to zebranie tych samych wyrazów różniących się wielkością pierwszej litery.

Listing Wybrane metody klasy FeatureExtractorPos

|  |
| --- |
| **def** stem\_and\_lemmatize(self, word, tag):  **return** self.lemmatizer.lemmatize(self.stemmer.stem(word), pos=self.get\_wordnet\_pos(tag))  **def** pos\_generator(self):  tagged\_words = self.tagger.tag(self.words)   **for** (word, tag) **in** tagged\_words:  **if** any(tag.startswith(prefix) **for** prefix **in** [**"NN"**, **"VB"**, **"JJ"**, **"RB"**]) **and not** self.is\_one\_sign(word) **and** \  **not** self.is\_in\_stopwords(word) **and** word.isalpha() **and not** self.is\_in\_tv\_set(word):  processed\_word = self.stem\_and\_lemmatize(word, tag)  **yield** processed\_word.lower() |

Słowa zwracane przez generator w przypadku binarnej macierzy cech umieszczane są w zbiorze z biblioteki standardowej języka Python. Takie działanie gwarantuje usunięcie powtórzeń. Następnie zbiór przekształca się w słownik z wartościami oznaczonymi na „True”. Kiedy klient programista chce stworzyć macierz zliczania cech, wyrazy po odfiltrowaniu tych nie spełniających przyjętych reguł, bezpośrednio umieszczane są w słowniku z jednoczesnym zliczaniem ilości ich wystąpień. Kolejnym zadaniem klasy FeaturePosExtractor jest znalezienie kolokacji. Do tego celu powstała klasa CollocationFinder umieszczona w komponencie „Przetwarzanie Tekstu”. Szukane są łańcuchy znaków składające się z dwóch i trzech słów występujące co najmniej dwa razy w artykule. Rysunek 9 przedstawia sposób działania klasy FeatureExtractorPos.



Rysunek Diagram maszyny stanowej - tworzenie wektora cech pojedynczego dokumentu

Warto zwrócić uwagę na listing 7, gdzie została pokazana klasa CollocationsFinder. Zawiera ona dwie podstawowe metody bigram\_finder i trigram\_finder. Każda z nich szuka wystąpień związków wyrazowych w zbiorze słów. Kolokacje nie mogą zawierać wyrazów składających się z mniej niż 3 znaków, znajdujących się w grupie stopwords oraz organizacji, z których stron internetowych pobierano dokumenty oraz znaków nie będących w alfabecie. Dodatkowo nałożono minimalne ograniczenie dotyczące częstości wystąpień w wysokości 2 na artykuł. Rezultatem obu metod są 4 najlepsze kolokacje według przyjętej miary asocjacji: funkcji prawdopodobieństwa.

Listing Wyszukiwanie kolokacji

|  |
| --- |
| **class** CollocationsFinder(object):   **def** \_\_init\_\_(self):  self.stopset = set(stopwords.words(**'english'**))  self.tv\_set = (**'fox'**, **'news'**, **'bbc'**, **'yahoo'**, **'telegraph'**, **'reuters'**)  self.filter\_stops = **lambda** w: len(w) < 3 **or** w **in** self.stopset **or** w **in** self.tv\_set   **def** filter\_numbers(self, words):  **return** [word.lower() **for** word **in** words **if** word.isalpha()]   **def** bigram\_finder(self, words):  tcf= BigramCollocationFinder.from\_words(  self.filter\_numbers(words})   tcf.apply\_word\_filter(self.filter\_stops)  tcf.apply\_freq\_filter(2)  **return** tcf.nbest(BigramAssocMeasures.likelihood\_ratio, 4)   **def** trigram\_finder(self, words):  tcf = TrigramCollocationFinder.from\_words(  self.filter\_numbers(words))  tcf.apply\_word\_filter(self.filter\_stops)  tcf.apply\_freq\_filter(2)  **return** tcf.nbest(TrigramAssocMeasures.likelihood\_ratio, 4) |

* 1. **Trenowanie modeli klasyfikatorów**

Wszystkie opisane wyżej komponenty miały na celu przygotowanie danych niezbędnych w procesie budowania modelu klasyfikatora w module „Trener”. Dokonano wyboru modeli klasyfikatorów. Są nimi Drzewa Decyzyjne, Bernoulli Naive Bayes, K Najbliższych Sąsiadów, Support Vector Machine oraz Drzewa Decyzyjne. Wymienione wcześniej algorytmy wykorzystano z biblioteki Scikit-Learn. Pakiet NLTK zawiera klasę SklearnClassifier stanowiącą opakowanie modeli klasyfikatorów. Listing 8 przedstawia fragment kodu źródłowego wyżej wymienionej klasy. W konstruktorze przyjmuje ona

Listing Fragment kodu źródłowego klasy SklearnClassifier[[4]](#footnote-5)

|  |
| --- |
| **class** SklearnClassifier(ClassifierI):   **def** \_\_init\_\_(self, estimator, dtype=float, sparse=True):   self.\_clf = estimator  self.\_encoder = LabelEncoder()  self.\_vectorizer = DictVectorizer(dtype=dtype, sparse=sparse)    **def** train(self, labeled\_featuresets):   X, y = list(compat.izip(\*labeled\_featuresets))  X = self.\_vectorizer.fit\_transform(X)  y = self.\_encoder.fit\_transform(y)  self.\_clf.fit(X, y)   **return** self |

obiekt algorytmu klasyfikującego z pakietu Scikit-Learn oraz typ danych wartości słownika wektorów cech. Kluczową metodę stanowi train. W ciele funkcji składowej znajduje się wykorzystanie metody fit\_transform klasy DictVectorizer do przekształcenia listy słowników na macierz biblioteki Numpy o dokładnie takiej samej reprezentacji jak ta opisana w rozdziale 2.2. Wyodrębniona zostaje ostatnia kolumna prezentująca klasy dokumentów. Funkcja fit przyjmuje przekształconą macierz i tworzy obiekt klasyfikatora. Listing 9 prezentuje fragment modułu „Trener”.

Listing Trenowanie modeli klasyfikatorów

|  |
| --- |
| **def** trainer(train\_documents, bag\_of\_words, classifier\_object, type, name):  **print**(**"Training {0} ..."**.format(name))  train\_set = apply\_features(bag\_of\_words, train\_documents)   classifier = SklearnClassifier(classifier\_object, type)  classifier.train(train\_set)   **with** open(**"Classifiers/"** + name + **".pickle"**, **'wb'**) **as** file\_handler:  dump(classifier, file\_handler)  **print**(**"Done"**)  **print**(**"Done"**)  algorithms = [DecisionTreeClassifier(), KNeighborsClassifier(), LinearSVC(), LogisticRegression(), BernoulliNB()]  **for** algorithm **in** algorithms:  class\_name = algorithm.\_\_class\_\_.\_\_name\_\_  trainer(train\_documents, binary\_bag\_of\_words,  deepcopy(algorithm), bool, class\_name + **"\_bool"**)  trainer(train\_documents, counted\_bag\_of\_words,  deepcopy(algorithm), int, class\_name + **"\_int"**)  trainer(train\_documents, counted\_bag\_of\_words,  Pipeline([(**'tfidf'**, TfidfTransformer()),  (**'nb'**, deepcopy(algorithm))]), float, class\_name + **"\_tfidf"**) |

Funkcja trainer przyjmuje zbiór trenujący w postaci listy dwuelementowych krotek, gdzie pierwszym elementem jest liczba słów dokumentu, a drugim jego klasa, funkcję tworzącą listę wektorów cech, typ wartości kluczy słownika zbiorów cech i nazwę klasyfikatora. W procesie uczenia bierze udział klasa SklearnClassifier. Model klasyfikatora zostaje zserializowany i zapisany do pliku w celu późniejszego wykorzystania. Dla każdego algorytmu powstają 3 modele różniące się sposobem reprezentacji macierzy. Do przekształcenia macierzy zliczania cech algorytmem tfidf użyto klasę TfidfTransformer z pakietu Scikit-Learn.

Klasyfikatory uczono z parametrami przedstawionymi na listingu 10.

Listing Parametry wybranych klasyfikatorów podczas procesu uczenia

|  |
| --- |
| sklearn.tree.**DecisionTreeClassifier**(criterion='gini', splitter='best', max\_depth=None, min\_samples\_split=2,min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None,class\_weight=None, presort=False)  sklearn.neighbors.**KNeighborsClassifier**(n\_neighbors=5, weights='uniform', algorithm='auto', leaf\_size=30,p=2, metric='minkowski', metric\_params=None, n\_jobs=1, \*\*kwargs)  sklearn.linear\_model.**LogisticRegression**(penalty='l2', dual=False, tol=0.0001, C=1.0, fit\_intercept=True,intercept\_scaling=1, class\_weight=None, random\_state=None, solver='liblinear', max\_iter=100, multi\_class='ovr',verbose=0, warm\_start=False, n\_jobs=1)  sklearn.svm.**LinearSVC**(penalty='l2', loss='squared\_hinge', dual=True, tol=0.0001, C=1.0, multi\_class='ovr',fit\_intercept=True, intercept\_scaling=1, class\_weight=None, verbose=0, random\_state=None, max\_iter=1000)  sklearn.naive\_bayes.**BernoulliNB**(alpha=1.0, fit\_prior=True, class\_prior=None) |

Parametr criterion algorytmu Drzewa Decyzyjne oznacza funkcję pomiaru jakości podziału. Wybrano kryterium Gini Impurtiy. Split mówi o strategii podziału każdego węzła. Wartość best oznacza najlepszy podział. Oprócz niej dostępna jest jeszcze „random”, czyli najlepszy losowy podział. Max\_features znaczy liczbę cech do rozważenia podczas poszukiwania najlepszego miejsca podziału. Jeżeli na tej pozycji zostanie podany None, algorytm weźmie pod uwagę wszystkie cechy. Max\_depth to maksymalna głębokość drzewa. Jeżeli w tym parametrze został ustawiony None to węzły są rozwijane dopóki liście staną się czyste lub wszystkie liście zawierają mniej próbek niż wartość min\_samples\_split, która oznacza minimalną liczbę próbek wymaganą do podziału wewnętrznego liścia. Min\_samples\_leaf reprezentuje minimalną liczbę próbek wymaganą do powstania liścia, min\_weight\_fraction minimalny ważony ułamek próbek wejściowych wymaganych do powstania liścia, a max\_leaf\_nodes ogranicza liczbę liści. Class\_weight służy do nadania wag klasom, gdy podano opcję None wszystkie etykiety mają wagę 1. Presort przyjmuje zmienną typu bool informującą o tym, czy posortować dane w celu przyspieszenia ustalania najlepszych miejsc podziału. Parametr random\_state dotyczy wyboru generatora liczb losowych.[[5]](#footnote-6)

W przypadku trenowania modelu K Najbliższych Sąsiadów n\_neighbours reprezentuje liczbę sąsiadów wykorzystywaną do zapytań algorytmu. Weight oznacza funkcję wag używaną do przewidywania. Przyjęty parametr „uniform” powoduje, że wszystkie punkty w każdym sąsiedztwie są ważone jednakowo. Algorithm znaczy metodę do obliczania najbliższych sąsiadów. Wybrana opcja „auto” prowadzi do wyboru najbardziej odpowiedniego algorytmu ( jednego z Brute Force, Ball Tree, KD Tree ) na podstawie podanej macierzy cech. Leaf size mówi rozmiarze liści, gdy zostanie wybrana metoda oparta na drzewie. Parametr metric służy do wyspecyfikowania metryki użytej do obliczania odległości w drzewie. P oznacza wartość współczynnika mocy wybranej miary „Minkowski”. N\_Jobs specyfikuje liczbę równoległych zadań jaką należy uruchomić podczas poszukiwania sąsiadów.[[6]](#footnote-7)

Najwięcej parametrów przyjmuje algorytm Regresji Logistycznej. Penalty oznacza normę przyjętą w procesie penalizacji, loss określa funkcję strat, tol kryterium stopu, a C odwrotność parametru regularyzacji. Intercept scalling określa czy stała powinna być dodawana do funkcji decyzji. Solver reprezentuje algorytm wybrany do optymalizacji funkcji przewidującej klasę, a max\_iter mówi o liczbie jego iteracji. Class\_weight podobnie jak w przypadku Drzew Decyzyjnych służy do nadania wag klasom, a random state dotyczy wyboru generatora liczb losowych. Multi\_class, verbose, warm\_start są argumentami przekazywanymi do algorytmu solver i zależą od tego jaką metodę zdecydowano wykorzystać[[7]](#footnote-8). Odpowiedniki parametrów metody Regresji Logistycznej przyjmuje model klasyfikatora oparty o algorytm Support Vector Machine z liniowym jądrem.

Ostatnia z rozważanych metod Bernoulli Naive Bayes posiada 3 argumenty. Pierwszy z nich alpha oznacza dodatkową stałą „wygładzającą” Laplace’a, drugi fit\_prior mówi czy zmieniać wcześniejszym klasom prawdopodobieństwo podczas uczenia, a trzeci class\_prior określa z góry prawdopodobieństwo klas, niezależnie od podanej macierzy cech.[[8]](#footnote-9)

1. **Badania skuteczności klasyfikacji oraz rozmiaru macierzy cech**

W tym rozdziale zostaną przedstawione miary dotyczące oceny jakości klasyfikatorów oraz wyniki testów skuteczności klasyfikacji. Z przestrzeni o liczebności 1254 dokumentów stworzono zbiór testowy, który wykorzystano do przeprowadzenia badań. Z każdej klasy w procesie testowania wzięło udział 25 % artykułów. W eksperymencie wzięto pod uwagę, także zbiór uczący. Został on wykorzystany do sprawdzenia jak zmieniała się wymiarowość macierzy cech po zastosowaniu metod ograniczających jej rozmiar. Oba zbiory stanowiły zbiory rozłączne. Artykuły, które brały udział w procesie trenowania modeli opartych o algorytmy uczące nie brały udziału w testach jakości klasyfikatorów. Cała przestrzeń została wykorzystana. W tabeli 1 przedstawiono informację ile dokumentów znalazło się w poszczególnych zbiorach.

Tabela Rozmiar zbiorów tekstów wyznaczonych z przestrzeni dokumentów

|  |  |  |  |
| --- | --- | --- | --- |
| **Rozmiar zbiorów tekstów wyznaczonych z przestrzeni dokumentów** | | | |
| **Klasa** | **Liczba dokumentów w przestrzeni** | **Liczba dokumentów w zbiorze trenującym** | **Liczba dokumentów w zbiorze testowym** |
| SPORT | 433 | 317 | 116 |
| POLITYKA | 357 | 267 | 90 |
| ZDROWIE | 464 | 355 | 109 |
| Wszystkie klasy | 1254 | 939 | 315 |

W zbiorze trenującym znalazło się 939 dokumentów, natomiast w zbiorze testowym 315. Najbardziej liczną klasą zostało „Zdrowie” z 465 artykułami, a najmniej „Polityka” z 357 tekstami. Do wykonania pomiarów wykorzystano funkcje accurancy, precision, recall, f\_measure zawarte w module NLTK w pakiecie metrics.

* 1. **Miary skuteczności klasyfikacji**

W celu wykonania eksperymentu zostały wybrane miary oceniające skuteczność nauczonych modeli. Pierwszą z nich jest trafność klasyfikacji (ang. accurancy), nazywana także dokładnością . Termin pojawił się w serii publikacji standardu ISO 5725 w 1994 roku. Określa zbliżenie pomiaru do wartości rzeczywistej[[9]](#footnote-10). Wyrażany jest wzorem

gdzie Np oznacza liczbę poprawnie sklasyfikowanych próbek testowych, Nt  całkowitą liczbę próbek. Dokładność klasyfikatora wyraża się dla danego zbioru testowego. Wartość może różnić się w zależność od doboru przykładów testowych.

Oprócz trafności istnieją inne metryki służące do oceny klasyfikatora. Podczas badań zostały wykorzystane dwie najbardziej popularne precyzja (ang. precision) i wrażliwość ( ang. recall ). Podane wyżej miary najczęściej stosuje się w klasyfikacji binarnej. W przypadku rozważanych w niniejszej pracy klasyfikatorów wieloklasowych precyzja i wrażliwość liczona jest oddzielnie dla każdej z kategorii. W celu zrozumienia tych dwóch metryk należy przytoczyć pojęcia wyników fałszywie dodatnich (ang. false positive) i fałszywie ujemnych (ang. false negative). Wynik fałszywie dodatni pojawia się kiedy klasyfikator przydziela zbiorowi cech tekstu etykietę, której nie powinien. Wynik fałszywie ujemny występuje, gdy klasyfikator nie przypisuje właściwej kategorii dla zbioru cech do którego powinien. Przykładowo dla etykiety „Sport” wynikami fałszywie dodatnimi będą wszystkie dokumenty z klas „Polityka” i „Zdrowie” błędnie skalsyfikowane jako „Sport”. Wynikami fałszywie ujemnymi etykiety „Sport” zostaną wszystkie artykuły z tej klasy błędnie skalsyfikowane do pozostałych kategorii. Analogicznie, wynikami prawdziwie dodatnimi klasy „Sport” będą wszystkie prawidłowo sklasyfikowane próbki testowe tej etykiety. Wynikami prawdziwie ujemnymi zostaną określone teksty innych kategorii prawidłowo nie przydzielonych do etykiety „Sport”. Powyższy przykład przedstawiono w tabeli 2.

Tabela Wyniki prawdziwe i fałszywe kategorii sport

|  |  |  |
| --- | --- | --- |
| Oryginalne klasy | Przewidywane klasy | |
| SPORT | Pozostałe klasy |
| SPORT | Liczba wyników prawdziwie ujemnych | Liczba wyników fałszywie ujemnych |
| Pozostałe klasy | Liczba wyników fałszywie dodatnich | Liczba wyników prawdziwie dodatnich |

Precyzja wyrażona jest wzorem

gdzie PD oznacza liczbę wyników prawdziwie dodatnich, natomiast FD liczbę wyników fałszywie dodatnich. Precyzja jest reprezentacją braku wyników fałszywie dodatnich.

Wrażliwość wyrażona jest wzorem

gdzie PD oznacza taką samą miarę jak w przypadku precyzji, a FU liczbę wyników fałszywie ujemnych. Wrażliwość informuje o braku wyników fałszywie ujemnych. Wskaźniki precyzji i wrażliwości często rywalizują ze sobą. Im większa jest precyzja tym mniejsza wrażliwość i odwrotnie.[[10]](#footnote-11)

Ostatnią metrykę wykorzystaną w badaniach określa się terminem „miary F” (ang „F-measure”). Miara określa średnią harmoniczną pomiędzy precyzją i wrażliwością. W testach skuteczności klasyfikacji często wykorzystywana jest jako alternatywa dla dokładności.[[11]](#footnote-12) Jeżeli precyzja i wrażliwość będą bardzo małe zostanie to odzwierciedlone w mierze F, ale niekoniecznie w trafności. Miara F wyrażona jest wzorem

gdzie oznacza precyzję, wrażliwość, a stałą zawierającą się w przedziale od 0 do 1. W badaniach przyjęto, że wynosi 0.5.

Podczas analizy wyników eksperymentu wykorzystano dla każdego algorytmu uczącego macierz pomyłek (ang. confusion matrix). Macierz reprezentuje tabela, w której każda komórka [i,j] informuje jak często etykieta j została przewidziana przez model klasyfikatora, gdy prawdziwą klasą była i. Komórki leżące na głównej przekątnej macierzy oznaczają klasy dobrze przewidziane, natomiast wszystkie inne mówią o błędach.

* 1. **Wpływ zastosowania metod ograniczania wymiarowości na rozmiar macierzy cech**

Pierwsze badanie dotyczyło sprawdzenia jak zmienia się rozmiar macierzy cech podczas uruchomiania i dokładania do poprzednich kolejno funkcji mających na celu ograniczenie jej wymiarowości. Wyniki zostały przedstawione w tabeli 3.

Tabela Zmiana rozmiaru macierzy cech

|  |  |  |
| --- | --- | --- |
| **Kolejno włączana funkcjonalność powodująca organicznie wymiarowości wektorów cech** | **Rozmiar macierzy cech** | |
|  | **X** | **Y** |
| Bag of words | 37808 | 939 |
| Wykluczenie słów składających się z jednego znaku | 37733 | 939 |
| Zebranie słów różniących się wielkością pierwszej litery | 33924 | 939 |
| Wykluczenie słów składających się ze znaków, których nie ma w alfabecie | 26501 | 939 |
| Wykluczenie słów należących do grupy stopwords dostępnej w bibliotece NLTK | 26481 | 939 |
| Wykluczenie słów należących do grupy nazw własnych organizacji, z których były pobierane dokumenty uczące | 26475 | 939 |
| Znakowanie części mowy (ang. POS-tagging), pozostawienie rzeczowników, czasowników, przymiotników i przysłówków | 26430 | 939 |
| Stemming, analiza słowotwórcza | 18904 | 939 |
| Lematyzacja - sprowadzenie formy fleksyjnej do postaci słownikowej | 18840 | 939 |

Do eksperymentu został wykorzystany zbiór trenujący o liczebności 939 dokumentów, stanowiący stały rozmiar wierszy macierzy. Test rozpoczęto od metody bag of words, czyli stworzenia wektorów cech na podstawie zbioru wszystkich słów w zbiorze trenującym Słowa stały się rozmiarem kolumn macierzy. Początkowo było ich 37808. Po wspólnym uruchomieniu wszystkich metod wyboru cech zostało ich 18840. Liczba komórek macierzy z 35 501 712 zmalała do 17 690 760. Wymiar wierszy zmniejszono o 50,2 %. Odnotowano zaskakująco niski wpływ metody znakowania części mowy na ograniczenie wymiaru kolumn. Po włączeniu tej metody zostało usuniętych 45 słów w stosunku do wcześniej uruchamianych metod. Największy wpływ na zmniejszenie liczby słów miał stemming. Dzięki tej funkcji zostało usuniętych 7526 słów.

* 1. **Porównanie skuteczności klasyfikatorów**

W tym teście zostały wyliczone miary przedstawione w rozdziale 4.1 dla modeli klasyfikatorów wykorzystujących pięć algorytmów: Bernoulli Naive Bayes, Support Vector Machine, Drzewa Decyzyjne, K Najbliższych Sąsiadów i Regresja Logistyczna. Klasyfikatory były uczone macierzami cech występującymi w trzech reprezentacjach: z zastosowaniem zliczania słów, optymalizowanej metodami tfidf oraz binarnej. Do badań wykorzystano zbiór testowy przedstawiono tabeli 1. Wyniki trafności poszczególnych klasyfikatorów ilustruje tabela 4. Najbardziej dokładne wyniki zostały oznaczone kolorem zielonym, natomiast najmniej dokładne kolorem czerwonym.

Tabela Porównanie trafności modeli klasyfikatorów

|  |  |  |
| --- | --- | --- |
| **Porównanie trafności klasyfikatorów** | | |
| **Algorytm uczący** | **Reprezentacja macierzy cech** | **Trafność** |
|  |  | **[%]** |
| Bernoulli Naive Bayes | Binarna | 88,888 |
| Bernoulli Naive Bayes | Zliczanie wystąpień słów | 88,888 |
| Bernoulli Naive Bayes | Optymalizacja TFIDF | 88,888 |
| Drzewa Decyzyjne | Binarna | 87,619 |
| Drzewa Decyzyjne | Zliczanie wystąpień słów | 87,301 |
| Drzewa Decyzyjne | Optymalizacja TFIDF | 86,031 |
| K Najbliższych Sąsiadów | Binarna | 72,063 |
| K Najbliższych Sąsiadów | Zliczanie wystąpień słów | 67,619 |
| K Najbliższych Sąsiadów | Optymalizacja TFIDF | 94,603 |
| Regresja logistyczna | Binarna | 96,825 |
| Regresja logistyczna | Zliczanie wystąpień słów | 96,507 |
| Regresja logistyczna | Optymalizacja TFIDF | 96,190 |
| Support Vector Machine | Binarna | 95,555 |
| Support Vector Machine | Zliczanie wystąpień słów | 96,507 |
| Support Vector Machine | Optymalizacja TFIDF | 96,507 |

Najbardziej skutecznymi klasyfikatorami okazały się modele wykorzystujące algorytmy Support Vector Machine i Regresja Logistyczna osiągając wyniki powyżej 95 %. Najsłabiej wypadły klasyfikatory oparte o metody K Najbliższych Sąsiadów uczone macierzami liczebności słów i binarnymi. Jednak metoda K Najbliższych Sąsiadów połączona z optymalizacją tfidf macierzy dała wynik 94,603 % o 22,54 % lepszy od tego samego algorytmu uczonego macierzą binarną, a o 26,948 % od trenowanego macierzą liczebności słów. Pozostałe algorytmy przekroczyły próg 85 % dokładności. Poza przykładem algorytmu K Najbliższych Sąsiadów dla wykorzystanego zbioru trenującego trudno znaleźć jednoznaczną odpowiedź jak wpływa reprezentacja macierzy cech na dokładność klasyfikacji. W przypadku klasyfikatora opartego o algorytm Bernoulli Naive Bayes, model klasyfikatora zawsze sprowadzał postać macierzy do binarnej, dlatego wyniki są identyczne. W przypadku Support Vector Machine najlepsze wyniki 96,507 % osiągnęły modele uczone macierzami liczebności słów i te optymalizowane metodą tfidf. Przewaga nad trafnością modelu uczonego macierzą binarną była niewielka 0,956 %. Przykłady Drzew Decyzyjnych i Regresji Liniowej pokazują, że tutaj najlepiej sprawdziły się klasyfikatory uczone macierzą binarną, jednak analogicznie do przykładu Support Vector Machine przewaga była niewielka i wyniosła ok. 1 %. Warto zaznaczyć, że są to wyniki dla wybranego zbioru trenującego. Jeżeli zostałby zmieniony zbiór, trafność także zmieniłaby się.

W tabelach 5, 6, 7 zostały przedstawione kolejno wyniki dotyczące precyzji, wrażliwości oraz miary F, poszczególnych klas klasyfikatorów. Kolorem czerwonym zostały oznaczone najsłabsze wyniki dla danej klasy, a kolorem zielonym najlepsze.

Tabela Porównanie precyzji klas klasyfikatorów

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Porównanie precyzji klas modeli klasyfikatorów** | | | | |
| **Algorytm uczący** | **Reprezentacja macierzy cech** | **Precyzja** | | |
|  |  | **Zbiór klas klasyfikatora** | | |
| **SPORT** | **POLITYKA** | **ZDROWIE** |
| **[%]** | **[%]** | **[%]** |
| Bernoulli Naive Bayes | Binarna | 80,281 | 95,714 | 96,116 |
| Bernoulli Naive Bayes | Zliczanie wystąpień słów | 80,281 | 95,714 | 96,116 |
| Bernoulli Naive Bayes | Optymalizacja TFIDF | 80,281 | 95,714 | 96,116 |
| Drzewa Decyzyjne | Binarna | 88,888 | 85,882 | 87,500 |
| Drzewa Decyzyjne | Zliczanie wystąpień słów | 89,682 | 84,523 | 86,666 |
| Drzewa Decyzyjne | Optymalizacja TFIDF | 86,900 | 78,651 | 88,118 |
| K Najbliższych Sąsiadów | Binarna | 100,000 | 51,497 | 90,277 |
| K Najbliższych Sąsiadów | Zliczanie wystąpień słów | 100,000 | 47,567 | 91,935 |
| K Najbliższych Sąsiadów | Optymalizacja TFIDF | 96,581 | 91,304 | 95,283 |
| Regresja logistyczna | Binarna | 95,867 | 97,701 | 97,196 |
| Regresja logistyczna | Zliczanie wystąpień słów | 94,262 | 97,647 | 98,148 |
| Regresja logistyczna | Optymalizacja TFIDF | 95,041 | 97,674 | 96,296 |
| Support Vector Machine | Binarna | 96,666 | 94,318 | 95,327 |
| Support Vector Machine | Zliczanie wystąpień słów | 95,041 | 97,647 | 97,247 |
| Support Vector Machine | Optymalizacja TFIDF | 98,275 | 94,444 | 96,330 |

Tabela Porównanie wrażliwości klas modeli klasyfikatorów

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Porównanie wrażliwości klas modeli klasyfikatorów** | | | | |
| **Algorytm uczący** | **Reprezentacja macierzy cech** | **Wrażliwość** | | |
|  |  | **Zbiór klas klasyfikatora** | | |
| **SPORT** | **POLITYKA** | **ZDROWIE** |
| **[%]** | **[%]** | **[%]** |
| Bernoulli Naive Bayes | Binarna | 98,275 | 74,444 | 90,825 |
| Bernoulli Naive Bayes | Zliczanie wystąpień słów | 98,275 | 74,444 | 90,825 |
| Bernoulli Naive Bayes | Optymalizacja TFIDF | 98,275 | 74,444 | 90,825 |
| Drzewa Decyzyjne | Binarna | 83,486 | 81,111 | 96,551 |
| Drzewa Decyzyjne | Zliczanie wystąpień słów | 97,413 | 78,888 | 83,486 |
| Drzewa Decyzyjne | Optymalizacja TFIDF | 96,551 | 77,777 | 81,651 |
| K Najbliższych Sąsiadów | Binarna | 65,517 | 95,555 | 59,633 |
| K Najbliższych Sąsiadów | Zliczanie wystąpień słów | 58,620 | 97,777 | 52,293 |
| K Najbliższych Sąsiadów | Optymalizacja TFIDF | 97,413 | 93,333 | 92,660 |
| Regresja logistyczna | Binarna | 100,000 | 94,444 | 95,412 |
| Regresja logistyczna | Zliczanie wystąpień słów | 99,137 | 92,222 | 97,247 |
| Regresja logistyczna | Optymalizacja TFIDF | 99,137 | 93,333 | 95,412 |
| Support Vector Machine | Binarna | 100,000 | 92,222 | 93,577 |
| Support Vector Machine | Zliczanie wystąpień słów | 99,137 | 92,222 | 97,247 |
| Support Vector Machine | Optymalizacja TFIDF | 96,610 | 96,590 | 96,330 |

Tabela Porównanie miary F klas modeli klasyfikatorów

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Porównanie miary F klas modeli klasyfikatorów** | | | | | |
| **Algorytm uczący** | **Reprezentacja macierzy cech** | **Miara F** | | | |
|  |  | **Zbiór klas klasyfikatora** | | | |
| **SPORT** | **POLITYKA** | **ZDROWIE** | **Średnia wszystkich klas** |
| **[%]** | **[%]** | **[%]** | **[%]** |
| Bernoulli Naive Bayes | Binarna | 88,372 | 83,750 | 93,396 | 88,506 |
| Bernoulli Naive Bayes | Zliczanie wystąpień słów | 88,372 | 83,750 | 93,396 | 88,506 |
| Bernoulli Naive Bayes | Optymalizacja TFIDF | 88,372 | 83,750 | 93,396 | 88,506 |
| Drzewa Decyzyjne | Binarna | 92,561 | 83,428 | 85,446 | 87,145 |
| Drzewa Decyzyjne | Zliczanie wystąpień słów | 93,388 | 81,609 | 85,047 | 86,681 |
| Drzewa Decyzyjne | Optymalizacja TFIDF | 92,946 | 78,212 | 84,761 | 85,306 |
| K Najbliższych Sąsiadów | Binarna | 71,823 | 66,926 | 79,166 | 72,638 |
| K Najbliższych Sąsiadów | Zliczanie wystąpień słów | 73,913 | 64,000 | 66,666 | 68,193 |
| K Najbliższych Sąsiadów | Optymalizacja TFIDF | 96,995 | 92,307 | 93,953 | 94,418 |
| Regresja logistyczna | Binarna | 97,890 | 96,045 | 96,296 | 96,744 |
| Regresja logistyczna | Zliczanie wystąpień słów | 96,638 | 94,857 | 97,695 | 96,397 |
| Regresja logistyczna | Optymalizacja TFIDF | 97,046 | 95,454 | 95,852 | 96,117 |
| Support Vector Machine | Binarna | 98,305 | 93,258 | 94,444 | 95,336 |
| Support Vector Machine | Zliczanie wystąpień słów | 97,046 | 94,857 | 97,247 | 96,383 |
| Support Vector Machine | Optymalizacja TFIDF | 97,435 | 97,435 | 96,330 | 97,067 |

Największą precyzję i wrażliwość można zaobserwować wśród klasyfikatorów opartych o algorytmy Regresji Logistycznej i Support Vector Machine. Wyniki dla każdej klasy wyniosły powyżej 90 %. Potwierdzają to dane dotyczące miary F. Najsłabszy wynik miary F Regresji Logistycznej wyniósł 94,857 % w klasie Polityka. Klasyfikator był uczony macierzą reprezentowaną przez zliczanie słów. Klasyfikator oparty o Support Vector Machine, także osiągnął najsłabszy wynik 93,258 % w klasie Polityka. Tym razem klasyfikator uczono macierzą binarną. Wpływ reprezentacji macierzy widoczny jest w modelu Drzew Decyzyjnych w tabeli 6. Klasyfikator uczony macierzą binarną w klasie Sport osiągnął 83,486 %, a w klasie Polityka 96,551 %, gdy w przypadku trenowania innymi sposobami reprezentacji macierzy wyniki w kategorii Sport wyniosły ponad 90 %, a w klasie Polityka mieszczą się w przedziale od 81 % do 84 %.

W przypadku klasyfikatorów o najwyższych wynikach nie udało się zaobserwować zjawiska konkurencji pomiędzy wrażliwością i precyzją. Doskonale widoczne jest ono pośród klasyfikatorów wykorzystujących algorytm K Najbliższych Sąsiadów. Tutaj dla klasyfikatora trenowanego macierzą binarną obserwujemy precyzję w klasie Sport na poziomie 100 %, natomiast wrażliwość już na poziomie 67 %. Odwrotna sytuacja występuje w klasie Polityka, wrażliwość plasuje się na poziomie 51 %, a precyzja ok. 96 %. Metryki precyzji i wrażliwości równoważy miara F. Podany wyżej klasyfikator K Najbliższych Sąsiadów osiągnął średnią miarę F wszystkich klas wynoszącą ok. 72 %. Warto zwrócić uwagę na model klasyfikatora oparty o algorytm Bernoulli Naive Bayes. Wyniki miary F poszczególnych klas odzwierciedlają tutaj liczbę artykułów w zbiorze testowym w tych kategoriach. Im większa liczba artykułów pośród próbek testowych danej etykiety tym większy wynik miary F klasyfikatora. Najmniej liczna klasa Polityka ze zbioru testowego w każdym z testowanych modeli osiągnęła najmniejszy wynik w porównaniu do innych etykiet. Najbardziej liczna kategoria Zdrowie osiągnęła największą wartość miary F tylko w 6 z 15 testowanych klasyfikatorów. W pozostałych modelach najlepsze wyniki osiągała kategoria Sport. Przewaga w stosunku do innych klas była niewielka, nie przekraczało 8 %.

Porównując wyniki średniej miary F z trafnością klasyfikatorów potwierdza się stwierdzenie, że miara F jest alternatywną metryką dokładności. Wyniki z tabeli 4 w porównaniu do ostatniej kolumny tabeli 7 różnią się do 1 %. Obie metryki potwierdzają skuteczność klasyfikatorów.

Podczas analizy wyników poszczególnych klasyfikatorów zostały wykorzystane macierze strat. W tabelach 8,9,10 zostały one przedstawione dla modeli opartych o algorytm Support Vector Machine. Każda tabela pokazuje wyniki klasyfikatora uczonego macierzą cech reprezentowaną w inny sposób.

Tabela Macierz strat SVM - reprezentacja macierzy binarna

|  |  |  |  |
| --- | --- | --- | --- |
| Algorytm uczący | | Support Vector Machine | |
| Reprezentacja macierzy | | Binarna | |
|  | ZDROWIE' | POLITYKA' | SPORT' |
| ZDROWIE | 102 | 5 | 2 |
| POLITYKA | 5 | 83 | 2 |
| SPORT | 0 | 0 | 116 |

Tabela Macierz strat SVM - reprezentacja macierzy zliczanie słów

|  |  |  |  |
| --- | --- | --- | --- |
| Algorytm uczący | | Support Vector Machine | |
| Reprezentacja macierzy | | Zliczanie wystąpień słów | |
|  | ZDROWIE' | POLITYKA' | SPORT' |
| ZDROWIE | 106 | 1 | 2 |
| POLITYKA | 3 | 83 | 4 |
| SPORT | 0 | 1 | 115 |

Tabela Macierz strat SVM - reprezentacja macierzy optymalizacja TFIDF

|  |  |  |  |
| --- | --- | --- | --- |
| Algorytm uczący | | Support Vector Machine | |
| Reprezentacja macierzy | | Optymalizacja TFIDF | |
|  | ZDROWIE' | POLITYKA' | SPORT' |
| ZDROWIE | 105 | 2 | 2 |
| POLITYKA | 3 | 85 | 2 |
| SPORT | 1 | 1 | 114 |

W tabeli 4 wyniki trafności modeli Support Vector Machine trenowanych macierzą zliczania słów i optymalizowaną metodą tfidf posiadały taką samą wartość, która wyniosła 96,507 %. Analiza macierzy strat pokazała, że identyczny wynik trafności nie oznacza, że modele dla danego zbioru testowego przewidują w każdym przypadku takie same etykiety. W przypadku klasy Polityka 2 artykuły zakwalifikowane prawidłowo przez klasyfikator Support Vector Machine wykorzystujący optymalizację tfidf zostały przez ten trenowany macierzą zliczania słów dopasowane do innej klasy. W drugą stronę, także wystąpiły tego rodzaju błędy dotyczące pozostałych klas. Najwięcej wyników pozytywnie dodatnich klasy Sport posiada klasyfikator uczony macierzą binarną, w klasie Polityka macierzą optymalizowaną metodą tfidf, natomiast w kategorii Zdrowie macierzą zliczania słów.

* 1. **Wpływ zmiany rozmiaru macierzy cech na skuteczność klasyfikacji**

Ostanie badanie polegało na sprawdzeniu jak zmieniają się wyniki trafności klasyfikatorów w zależności od rozmiarów binarnej macierzy cech. Kolejno uruchomiano i dokładano do poprzednich funkcje mające na celu ograniczanie wymiarowości binarnej macierzy. Powstałymi w ten sposób macierzami trenowano modele klasyfikatorów opartych na pięciu algorytmach Bernoulli Naive Bayes, Drzewa Decyzyjne, K Najbliższych Sąsiadów, Regresja Logistyczna. Każdy klasyfikator był trenowany zbiorem trenującym oraz testowany zbiorem testowym przedstawionymi w tabeli 1. Dla wspomnianych wyżej modeli zostały zmierzone wyniki trafności klasyfikacji, zebrane i przedstawione w tabeli 11. Kolorem niebieskim zostały oznaczone najwyższe wartości dokładności danego modelu, a czerwonym najniższe.

Tabela Porównanie klasyfikatorów w zależności od zmiany rozmiaru binarnej macierzy cech

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Porównanie trafności klasyfikatorów w zależności od zmiany rozmiaru binarnej macierzy cech** | | | | | | |
| **Kolejno włączana funkcjonalność powodująca ograniczenie wymiarowości wektorów cech** | **Wymiar X wektora cech** | **Trafność** | | | | |
|  | | **Bernoulli Naive Bayes** | **Drzewa Decyzyjne** | **K Najbliższych Sąsiadów** | **Regresja logistyczna** | **Support Vector Machine** |
| **[%]** | **[%]** | **[%]** | **[%]** | **[%]** |
| Bag of Words | 37808 | 86,666 | 86,032 | 55,555 | 96,190 | 95,873 |
| Wykluczenie słów składających się z jednego znaku | 37733 | 86,349 | 86,666 | 55,238 | 96,190 | 96,507 |
| Zebranie słów różniących się wielkością pierwszej litery | 33924 | 86,984 | 82,539 | 56,190 | 96,507 | 96,190 |
| Wykluczenie słów składających się ze znaków, których nie ma w alfabecie | 26501 | 87,301 | 84,444 | 67,936 | 95,873 | 96,507 |
| Wykluczenie słów należących do grupy stopwords dostępnej w bibliotece NLTK | 26481 | 87,619 | 85,396 | 60,952 | 97,142 | 96,825 |
| Wykluczenie słów należących do grupy nazw własnych organizacji, z których były pobierane dokumenty uczące | 26475 | 87,301 | 84,761 | 59,047 | 96,825 | 96,190 |
| Znakowanie części mowy (ang. POS-tagging), pozostawienie rzeczowników, czasowników, przymiotników i przysłówków | 26430 | 88,253 | 84,126 | 59,365 | 96,825 | 95,873 |
| Stemming, analiza słowotwórcza | 18904 | 88,571 | 84,761 | 70,158 | 96,507 | 96,190 |
| Lematyzacja - sprowadzenie formy fleksyjnej do postaci słownikowej | 18840 | 88,888 | 88,253 | 72,063 | 96,825 | 95,555 |

W przypadku algorytmów Bernoulli Naive Bayes i K Najbliższych Sąsiadów wraz z ograniczaniem macierzy cech rosła trafność klasyfikacji. Przy ograniczeniu wymiaru kolumn z 37808 do 18840 trafność modelu opartego o K Najbliższych sąsiadów wzrosła o 16,508 %, natomiast o Bernoulli Naive Bayes o 2,222 %. Klasyfikatory Drzew Decyzyjnych i Regresji Logistycznej, także po włączeniu wszystkich funkcji ograniczających osiągnęły odpowiednio o 2,224 % i 0,690 % lepszy wynik od sytuacji, gdy były uczone macierzą ze wszystkimi słowami ze zbioru uczącego. Jedynym klasyfikatorem, którego trafność spadła był ten oparty o Support Vector Machine. Spadek w stosunku do wartości początkowej wyniósł 0,318 %. Wartość trafności po ograniczeniu rozmiaru kolumn o ponad 50 %, w przypadku modeli, których dokładność wzrosła nie zawsze oznaczała najwyższą. Po wykluczeniu słów należących do grupy stopwords z biblioteki NLTK, algorytm Regresji Logistycznej uzyskał najwyższy wynik 97,142 %. W trakcie dalszego ograniczania macierzy cech trafność spadła. W tym samym momencie najlepszy wynik uzyskał algorytm Support Vector Machine 96,825 %, o 0,956 %, więcej od chwili, gdy został nauczony macierzą zawierającą wszystkie słowa ze zbioru trenującego. W kolejnych krokach ograniczania wymiarowości dokładność tego algorytmu spadała. Wyniki badań pokazały, że ograniczenie wymiarowości wektorów cech wpływa korzystnie na trafność modeli opartych o Regresję Logistyczną, K Najbliższych Sąsiadów, Bernoulli Naive Bayes oraz Drzewa Decyzyjne. W przypadku algorytmu Support Vector Machine ograniczenie wymiarowości macierzy cech ma znikomy ( zmiany nie większe niż 1,5 %) wpływ na dokładność klasyfikacji.

W tabeli 12 zostały przedstawione wyniki zmiany miary F poszczególnych klas klasyfikatora wykorzystującego algorytm Bernoulli Naive Bayes podczas ograniczania rozmiaru macierzy binarnej, którą następnie użyto do trenowania modelu. Kolorem niebieskim oznaczono największe wartości w danej klasie, natomiast kolorem czerwonym najmniejsze.

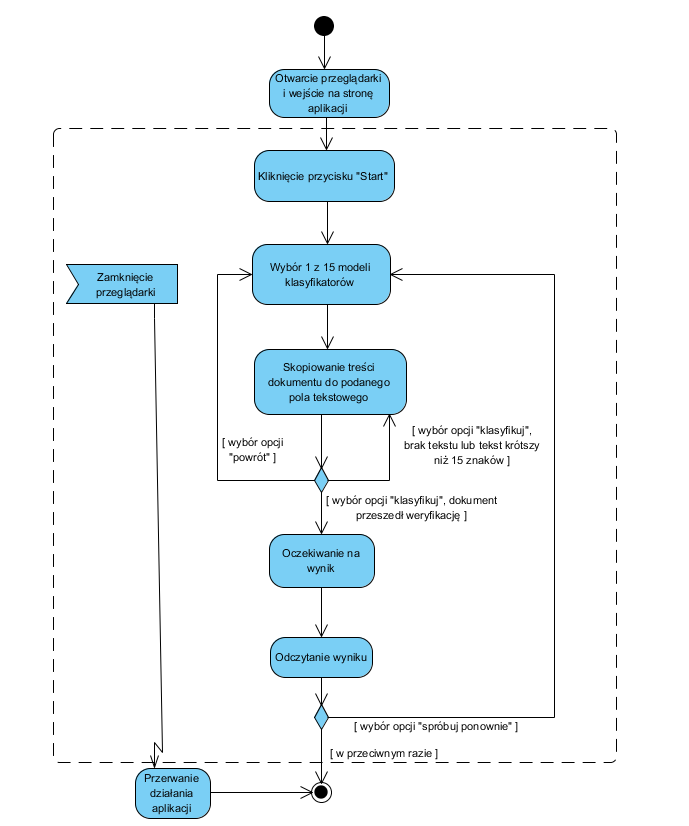
Tabela Wpływ ograniczenia wymiarowości binarnej macierzy cech na miarę f algorytmu Bernoulli Naive Bayes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Kolejno włączana funkcjonalność powodująca organicznie wymiarowości wektorów cech** | **Wymiar X wektora cech** | Miara F Bernoulli Naive Bayes | | | |
|  | | **SPORT** | **POLITYKA** | **ZDROWIE** | **Średnia wszystkich klas** |
| [%] | [%] | [%] | [%] |
| Bag of Words | 37808 | 86,142 | 79,220 | 92,822 | 86,061 |
| Wykluczenie słów składających się z jedngo znaku | 37733 | 85,820 | 79,220 | 92,308 | 85,783 |
| Zebranie słów różniących się wielkością pierwszej litery | 33924 | 86,466 | 79,220 | 93,333 | 86,340 |
| Wykluczenie słów składających się ze znaków, których nie ma w alfabecie | 26501 | 87,121 | 81,012 | 92,307 | 86,813 |
| Wykluczenie słów należących do grupy stopwords dostępnej w bibliotece NLTK | 26481 | 87,452 | 81,013 | 92,823 | 87,096 |
| Wykluczenie słów należących do grupy nazw własnych organizacji, z których były pobierane dokumenty uczące | 26475 | 87,452 | 80,503 | 92,308 | 86,754 |
| Znakowanie części mowy (ang. POS-tagging), pozostawienie rzeczowników, czasowników, przymiotników i przysłówków | 26430 | 87,786 | 82,278 | 93,333 | 87,799 |
| Stemming, analiza słowotwórcza | 18904 | 88,372 | 83,229 | 92,890 | 88,164 |
| Lematyzacja - sprowadzenie formy fleksyjnej do postaci słownikowej | 18840 | 88,372 | 83,750 | 93,396 | 88,506 |

Wyniki średniej miary F wszystkich klas metody Bernoulli Naive Bayes są zbliżone do wyników trafności tego algorytmu ( różnica odpowiadających pomiarów wynosi ok. 1 % ). Widoczna jest tutaj tendencja rosnąca. Najlepszy wynik osiągnięto po włączeniu wszystkich funkcji ograniczających. W momencie, gdy rozmiar macierzy cech osiągnął 18840 średnia miara F osiągnęła największą wartość 88,506 %. Najniższą wartość 86,061 % osiągnięto, gdy model został nauczony macierzą cech zawierającą wszystkie słowa ze zbioru testowego. Biorąc pod uwagę poszczególne klasy największe wartości miary F zostały osiągnięte podczas, gdy wymiarowość macierzy była najniższa. W przypadku algorytmu Bernoulli Naive Bayes ograniczanie wymiarowości macierzy cech miało dobry wpływ na skuteczność klasyfikacji. Wartość wyżej wymienionej metryki wraz z ograniczaniem wymiarowości wzrosła.

1. **Aplikacja Kliencka**

W ramach niniejszej pracy dyplomowej powstał interfejs webowy umożliwiający wykorzystanie powstałych modeli klasyfikatorów przez użytkownika w celu przyporządkowania klas dowolnym dokumentom. Aplikacja oparta jest na modelu klient-serwer składającym się z dwuwarstwowej architektury. Serwer aplikacji wykonano w technologii Python z wykorzystaniem modułu Django 1.8.5. Za pomocą biblioteki uwsgi połączono go z serwerem www Nginx. Po stronie klienta został wykorzystany framework Angular oraz html 5. Obie warstwy komunikują się za pomocą metod http przesyłając pliki w formacie json. Użytkownik korzysta z aplikacji poprzez przeglądarkę internetową. Sposób działania przedstawia diagram czynności na rysunku 10.



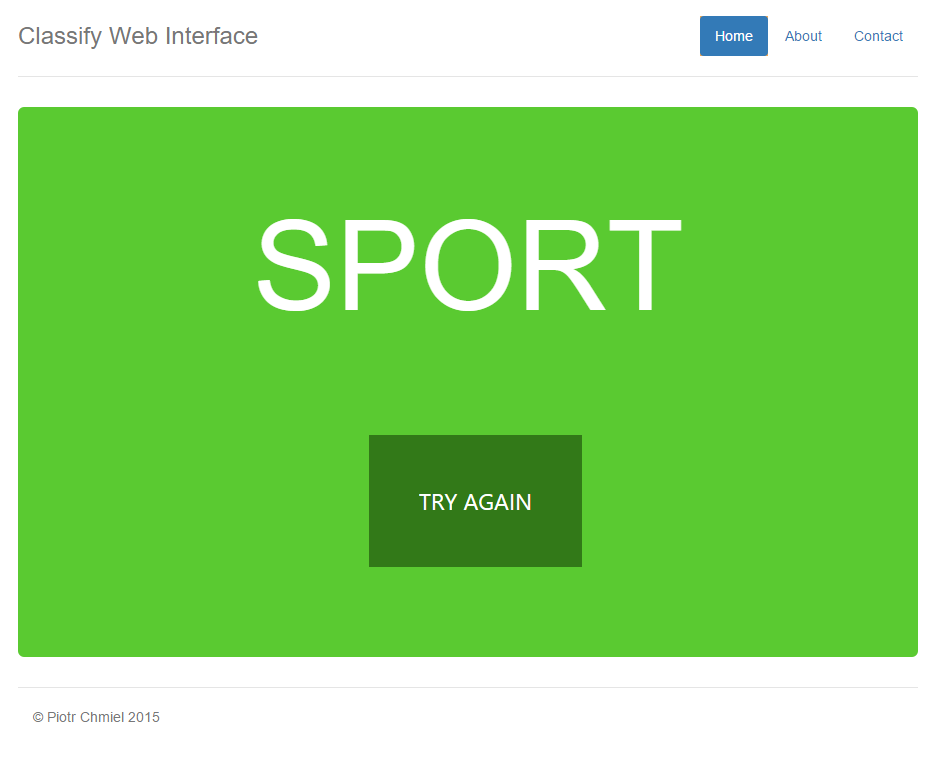
Rysunek Diagram czynności - sposób działania aplikacji klienckiej

Opisane w rozdziałach 3 i 4 modele klasyfikatorów zostały poddane serializacji, a następnie zapisane do pliku. W związku z tym użytkownik posiada do dyspozycji 5 różnych algorytmów klasyfikacji, każdy po 3 reprezentacje macierzy cech. Na listingu 11 został przedstawiony praktyczny sposób wykorzystania tych klasyfikatorów. W momencie, gdy aplikacja jest uruchamiana do pamięci w pliku konfiguracyjnym settings.py ładowane są modele. Użytkownik wybiera jeden z 15 nauczonych klasyfikatorów, później wkleja w pole tekstowe treść wybranego dokumentu. Kiedy nastąpi potwierdzenie i tekst przejdzie walidację zostaje wysłany metodą POST na serwer w celu przetworzenia i sklasyfikowania. Nagłówek zawiera dwie informacje: rodzaj modelu klasyfikatora i zawartość artykułu. Na podstawie nazwy przesłanego algorytmu funkcja dobiera odpowiednią metodę z komponentu „Generator Cech”, która tworzy macierz cech o wybranej reprezentacji. Zanim zostanie wykorzystana, tekst uprzednio dzieli się na słowa przy wykorzystaniu funkcji word\_tokenize z pakietu NLTK.

Listing Klasyfikacja tekstu - aplikacja kliencka

|  |
| --- |
| **class** ClassifyView(View):  **def** post(self, request, \*args, \*\*kwargs):  response = {}  **try**:  classifier = request.POST[**'classifier'**]  text = request.POST[**'text'**]  **except** Exception **as** e:  **print**(e)  response[**'status'**] = **"Error"** response[**'message'**] = **"Unexpected errror. Please Try Again !"  else**:  **if 'bool' in** classifier:  extractor = binary\_bag\_of\_words  **else**:  extractor = counted\_bag\_of\_words   **try**:  message = classify(text, self.get\_classifier(classifier),  extractor)  **if** message **is not** None:  response[**'status'**] = **"Success"** response[**'message'**] = message  **else**:  response[**'Error'**] = **"Error"** response[**'message'**] = **"Unable to classify text"  except** Exception **as** e:  **print**(e)  response[**'status'**] = **"Error"** response[**'message'**] = **"Unexpected error during**  **classification. Please Try Again !"   print**(response)  **return** JsonResponse(response)  **def** classify(content, classifier, bag\_of\_words\_feature\_extractor):  **try**:  **return** classifier.classify(bag\_of\_words\_feature\_extractor(  word\_tokenize(content)))  **except**:  **return** None |

Kolejnym krokiem jest wykorzystanie wbudowanej funkcji składowej classify każdego klasyfikatora do uzyskania klasy dokumentu. Jeżeli nie wystąpił błąd zostaje zwrócona klientowi odpowiedź w formacie json. W przeglądarce użytkownik otrzymuje graficzną prezentację wyniku klasyfikacji (rysunek 11). Po wciśnięciu przycisku try again może przejść cały proces dowolną ilość razy. W przypadku kiedy zostanie podany tekst nie przynależący do kategorii Sport, Polityka i Zdrowie zawsze zostanie zwrócona jedna z tych klas. Aplikacja zawiera dwie dodatkowe zakładki „About” i „Contact”. Zawierają one informacje o całym projekcie, użyte wersje bibliotek języka Python oraz dane kontaktowe twórcy.



Rysunek Aplikacja kliencka - rezultat klasyfikacji

1. **Podsumowanie**

**Bibliografia**

LITERATURA

Abu-Mostafa Y.S., Malik. M.I, Lin H. T., *Learning From Data*, AML Book, 2012.

Beazley D., Jones B.K., *Python Receptury*, tłum. T. Walczak, Helion, Gliwice 2014.

Beck K., Brant J., Fowler M., Roberts D., Opdyke W., *Refaktoryzacja – Ulepszanie struktury istniejącego Kodu*, tłum. J. Walkowska, Helion, Gliwice 2011

Bird S., Klein E., Loper. E., *Natural Language Processing with Python*, O’Reilly, Sebastopol 2011.

Bishop Ch., *Pattern Recognition and Machine Learning*, Springer, Berlin 2006.

Bowles M., *Machine Learning in Python: Essential Techniques for Predictive Analysis*, Wiley, New Jersey 2011.

Coelho L.P., Richert W., *Building Machine Learning Systems with Python Second Edition*, Packt Publishing Ltd., Birmingham 2015.

DasGupta A., *Probability for Statistics and Machine Learning*, Springer, Berlin 2013.

Friedman J., Hastie. T., Tibshirani R., *The Elements of Statistical Learning Data Mining, Inerference, and Prediction*, Springer, Berlin 2009.

Garreta R., Moncechi. G., *Learning scikit-learn: Machine Learning in Python*, Packt Publishing Ltd., Birmingham 2013.

Gorelick M., Ozsvald. I., *Python Programuj Szybko i Wydajnie*, tłum. P. Pilch, Helion, Gliwice 2015.

Kalbarczyk A., Kalbarczyk D., *AngularJS Pierwsze Kroki*, Helion, Gliwice 2015.

Manning Ch.D., Raghavan. P., Schütze. H. *Introduction to Information Retrieval*, Cambridge University Press, Cambridge 2008.

Martin R.C., *Czysty Kod – Podręcznik Dobrego Programisty*, tłum. P. Gonera, Helion, Gliwice 2010.

McKinney W., *Python for Data Analysis: Data Wrangling with Pandas, NumPy and IPython*, O’Reilly, Sebastopol 2012.

Mitchell R., *Web Scraping with Python: Collecting Data from the Modern Web*, O’Reilly, Sebastopol 2015.

Mohri M., Rostamizadeh A, Talwalkar. A. *Foundations of Machine Learning*, MIT Press, Cambridge MA USA 2012.

Murphy K. P. *Machine Learning: a Probabilistic Perspective* , MIT Press, Cambridge MA USA 2013.

Lutz M., *Python Wprowadzenie*, tłum A. Trojan, Helion, Gliwice 2009.

Layton R., *Learning Data Mining with Python*, Packt Publishing Ltd., Birmingham 2015.

Perkins J., *Python 3 Text Processing with NLTK 3 Cookbook*, Packt Publishing Ltd., Birmingham 2014.

Powers D., *Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation*, [w:] *„Journal of Machine Learning Technologies”*, red. Xiao-zhi Gao, Bioinfo Publications, nr. 2 2011.

Pustejovsky J. Stubbs A., *Natural Language Annotation for Machine Learning*, O’Reilly, Sebastopol 2012.

ŹRÓDŁA INTERNETOWE

*Dokumentacja Języka Python 3*, [online:] https://docs.python.org/3/, 10.08.2015.

*Dokumentacja Biblioteki AngularJS*, [online:] https://docs.angularjs.org/api, 01.11.2015.

*Dokumentacja Biblioteki Beautiful Soup*, [online:] http://www.crummy.com/software/BeautifulSoup/bs4/doc/ 11.11.2015.

*Dokumentacja Biblioteki Django*, [online:] https://docs.djangoproject.com/en/1.8/ , 01.11.2015.

*Dokumentacja Biblioteki Feed Parser*, [online:] https://pythonhosted.org/feedparser/ 11.11.2015.

*Dokumentacja Bibliotek Numpy i Scipy*, [online:] http://docs.scipy.org/doc/ 11.11.2015.

*Dokumentacja Biblioteki Natural Language Toolkit*, [online:] http://www.nltk.org/, 11.11.2015.

*Dokumentacja Biblioteki Scikit-Learn*, [online:] http://scikit-learn.org/stable/documentation.html, 11.11.2015.

*Dokumentacja Biblioteki uWsgi*, [online:] http://uwsgi-docs.readthedocs.org/en/latest/, 11.11.2015.

*Dokumentacja Serwera WWW Nginx*, [online:] http://nginx.org/en/docs/11.11.2015.

Jurafsky D., Manning Ch., *Natural Language Processing Course Lectures* [online:] *http://class.coursera.org/nlp/lecture*., 03.10.2015.

Norma ISO 5725-1:1994., *Accurancy (trueness and precision) of measurement and results – Part 1: General principles and definitions*, [online:] http://www.iso.org/iso/iso\_catalogue/catalogue\_tc/catalogue\_detail.htm?csnumber=11833, 03.10.2015.

*Total numer of Websites*, [online:] http://www.internetlivestats.com/total-number-of-websites/ 11.11.2015.

1. „Total numer of Websites”, http://www.internetlivestats.com/total-number-of-websites/ ( dostęp 15.11.2015 r. ) [↑](#footnote-ref-1)
2. S. Bird, E.Klein, E.Loper: „Natural Language Processing with Python”. Sebastopol 2009, O’Reilly, s. 222. [↑](#footnote-ref-3)
3. S. Bird, E.Klein, E.Loper: „Natural Language Processing with Python”. Sebastopol 2009, O’Reilly, s. 204. [↑](#footnote-ref-4)
4. Dokumentacja Biblioteki NLTK, http://www.nltk.org/\_modules/nltk/classify/scikitlearn.html ( dostęp 11.11.2015 r. ) [↑](#footnote-ref-5)
5. Dokumentacja klasy DecisionTree biblioteki Scikit-Learn, http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier (data dostępu 11.11.2015 r.) [↑](#footnote-ref-6)
6. Dokumentacja klasy K-Neighbors biblioteki Scikit-Learn, http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (data dostępu 11.11.2015 r.) [↑](#footnote-ref-7)
7. Dokumentacja klasy LogisticRegression biblioteki Scikit-Learn, http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html (data dostępu 11.11.2015 r.) [↑](#footnote-ref-8)
8. Dokumentacja klasy BernoulliNaiveBayes biblioteki Scikit-Learn, http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.BernoulliNB.html (data dostępu 11.11.2015 r. ) [↑](#footnote-ref-9)
9. BS ISO 5725-1: „Accuracy (trueness and precision) of measurement methods and results - Part 1: General principles and definitions.”, 1994, s. 1, [↑](#footnote-ref-10)
10. J. Perkins: „Python 3 Text Processing with NLTK 3 Cookbook”. Birmingham 2015, Packt Publishing LTD, s. 210. [↑](#footnote-ref-11)
11. D.Powers*:*„Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation”, w: „Journal of Machine Learning Technologies”, nr 2, 2011, s. 37-45  [↑](#footnote-ref-12)