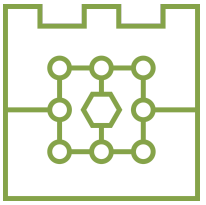
Politechnika Krakowska
im. Tadeusza Kościuszki
Wydział Informatyki i Telekomunikacji



Michał Gabryś

Numer albumu: 144 913

Wyszukiwanie podobieństw semantycznych w treściach zgłoszonych błędów testowanego oprogramowania stacji bazowych 5G.

Searching for semantic similarities in the content of reported bugs of 5G base station software under test.

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem:
dra inż. Radosława Kyci

Recenzent pracy:

Kraków 2023

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję wszystkim prowadzącym z Politechniki Krakowskiej, z którymi miałem przyjemność odbywać zajęcia oraz poznawać nowe rzeczy.

Spis treści

Streszczenie

W pracy opisano klasyfikację zgłoszenia o błędzie oprogramowania stacji bazowej 5G do potencjalnej grupy końcowej, która zajmie się rozwiązaniem. Badania odbywały się na danych pochodzących z organizacji Nokia Mobile Networks zajmującej się tworzeniem sprzętu 5G. Podczas badań wykorzystano techniki przetwarzania języka naturalnego oraz metody uczenia maszynowego, takie jak tokenizacja, normalizacja, wektoryzacja metodami TF-IDF i Word2Vec czy klastrowanie metodą DBSCAN, K-średnich lub GaussianMixtureModel. Do wizualizacji zostały wykorzystane t-SNE. Do porównania wyników użyto las losowy i podobieństwo cosinusowe. Badania rzuciły głębsze światło na problem wyszukiwania grupy końcowej oraz przybliżyły opracowanie optymalnego narzędzia, które w przyszłości ułatwi pracę testerom. Najlepsze wyniki dało połączenie wektoryzacji TF-IDF wraz z modelem lasu losowego.

Abstract

This thesis describes problem of classification software bug reports to company internal groups that should be able to take care of them. All data is reserved to Nokia Mobile Networks. Whole research consists of analyzing problem and dataset, applying various NLP techniques to it and measuring prediction's precision using machine learning models. Used techniques are: TF-IDF, Word2Vec, t-SNE, DBSCAN, K-means, GMM, random forest and cosine similiarity. Efforts described in this paper could make a significant progress in improving Nokia's workflow. The best result was obtained by combined approach in which TF-IDF with random forest were used.

1. Wstęp

1.1. Cel pracy

W dużych korporacjach, zatrudniających dziesiątki tysięcy pracowników, ważne jest uskutecznienie komunikacji między działami, aby zapewnić jak najszybsze rozwiązanie problemów i ukończenie testowania bez przeszkód. Jedną z fundamentalnych części tego procesu jest identyfikowanie i raportowanie błędów, na które natknęli się testerzy oraz współpracowanie z innymi działami w celu rozwiązania problemu. W firmie Nokia istnieje narzędzie, wykorzystywane do zgłaszania i naprawy błędów. Działanie takiego systemu nie jest idealne i będzie obciążone ludzkim błędem, co powoduje marnowanie pieniędzy oraz czasu. Przykładowo, niektóre zgłoszenia potrzebują nawet miesiąca, aby pokonać drogę od zgłoszenia do likwidacji problemu. Opóźnienie w analizie, spowodowane szukaniem eksperta, zdolnego do naprawienia błędu jest niepożądanym zjawiskiem i stanowi problem, który potencjalnie można zautomatyzować przy użyciu technik uczenia maszynowego i przetwarzania języka naturalnego.

Celem pracy jest poprawa jakości przydziału zgłoszeń do odpowiednich grup i tym samym usprawnienie całego systemu. Uogólniając, praca skupia się na rozwiązaniu problemu predykcji końcowej grupy za pomocą sztucznej inteligencji, Autor pracy przeprowadził badania w celu zmniejszenia czasu oraz zminimalizowania ludzkiego nadzoru, potrzebnego do optymalizacji całego procesu. W pracy opisane zostały próby wykorzystania sztucznej inteligencji przy rozwiązywaniu realnego problemu w firmie Nokia.

1.2. Zakres pracy

W pracy zawarto dogłębny przegląd problemu oraz opisano korzyści płynące z precyzyjnej predykcji końcowej grupy, do której powinno trafić poszczególne zgłoszenie. W zakresie badań przeprowadzonych w pracy znajdują się: odpowiednio udokumentowany zbiór danych, użyty do nauki modeli sztucznej inteligencji oraz opis metod oceny jakości tych modeli. Wykorzystanymi metodami wektoryzacji danych były: TF-IDF oraz Doc2Vec. Do redukcji wymiarowości posłużyło PCA i TruncatedSVD, a do obliczania predykcji zastosowano podobieństwo cosinusowe i las losowy. Użyto również metody DBSCAN, K-średnich oraz GaussianMixutreModel do klastrowania danych. Praca przedstawia również wyniki badań, które ilustrują skuteczność zastosowanego rozwiązania oraz analizę wyników w celu wyciągnięcia wniosków i zaleceń dla dalszych badań.

1.3. Metodyka pracy

Badania przeprowadzone na udostępnionym przez Nokię zbiorze danych odbywały się w następującej kolejności:

- przygotowanie, czyszczenie i filtrowanie danych,
- przetwarzanie i ekstrakcja danych tekstowych do postaci czytelnej dla modelu,
- walidacja spójności danych oraz zabezpieczenie się przed tzw. przeuczeniem
- porównanie modeli uczenia maszynowego pod kątem precyzji w predykcji

Powyższy potok przetwarzania danych pozwolił na pełne zrozumienie problemu, postawienie odpowiednich hipotez oraz zapewnienie spójności wyników. Do implementacji opisanych kroków został wykorzystany język programowania Python, łącznie z jego bibliotekami *pandas, numpy, matplotlib* do obróbki i wizualizacji danych. W przypadku algorytmów redukcji danych i modeli uczenia maszynowego, użyto *sci-kit* learn. Całość kodu była kompilowana za pomocą środowiska *Jupyter Notebook.* Wszystkie badania zostały przeprowadzone na serwerze zdalnym, posiadającym kartę graficzną NVIDIA GeForce RTX 2070 SUPER o pamięci wewnętrznej 8GB. Praca została podzielona na dwie części; teoretyczną i praktyczną. W części teoretycznej został opisany użyty potok przetwarzania języka naturalnego, a oprócz tego metody wektoryzacji, wizualizacji oraz wybrane zagadnienia wyszukiwania semantycznego. Część praktyczna składa się z przedstawienia problemu i nawiązania do modelu biznesowego w firmie. W tej części, autor pracy zaprezentował różne podejścia do badania związku pomiędzy zgłoszeniami testerskimi a końcowymi grupami oraz dokonał implementacji omawianych etapów i wizualizacji otrzymanych wyników.

2. Część teoretyczna

2.1. Zmienne ilościowe i jakościowe

W programowaniu zmienna odnosi się do adresu w pamięci komputera i może przyjąć wartości numeryczne, binarne lub tekstowe. W terminologii uczenia maszynowego zmiennymi będą cechy próbek znajdujących się w zbiorze danych. Gdy mamy do czynienia ze zmiennymi ilościowymi (numerycznymi) to mówimy że pochodzą one z rozkładu ciągłego. Z rozsądną dokładnością można założyć, że zmienna przechowywująca wzrost losowo wybranego człowieka żyjącego w Europie wyrażona w centymetrach jest zmienną typu ciągłego. Zakres jej wartości z pewnością mieści się w przedziale (0, 300)[1]. Zmienne ilościowe są najczęściej spotykane w problemach związanych z giełdą i ekonomią (przewidywanie wzrostów akcji) czy rynkami samochodowymi lub mieszkaniowymi (szacowanie cen w zależności od położenia i innych czynników). Można policzyć ich średnią, odchylenie standardowe czy amplitudę. Najczęściej używanymi modelami do zbiorów ze zmiennymi ilościowymi są modele regresji, na przykład liniowej.

Zmienne jakościowe (kategoryczne) są zazwyczaj wynikiem obserwacji lub wywiadów i opierają się na danych empirycznych. Mogą mieć charakter dwumianowy (binarny) i być:

- nominalne - zmienne jakościowe nieuporządkowane (rodzaj marki),
- uporządkowane - będace na pewnym poziomie, etapie (wykształcenie)[2].

Wtedy mówimy, że zmienna pochodzi z rozkładu dyskretnego. Nie może ona przyjąć wartości pomiędzy dwoma wartościami kategorycznymi - książka nie może zostać otwarta na stronie 332,5[1]. Problemy związane z klasyfikacją kategorii to na przykład wykrywanie spamu czy analiza wydźwięku (pozytywny czy negatywny).

2.2. Pomiary skuteczności modelu

Gotowe rozwiązanie dające predykcje to nie wszystko. W procesie rozwoju sztucznej inteligencji, bardzo znaczące jest obliczenie metryk reprezentujących jakość wyników. Musi to zostać zrobione na

parę sposobów, aby upewnić się że model nie będzie wprowadzał w błąd jego użytkowników. Na początku, trenowanie i testowanie może odbyć się za pomocą walidacji krzyżowej, która polega na losowym podziale zbioru danych na dowolną liczbę podzbiorów. Następnie, jeden podzbiór zostaje wykorzystany do testowania, a pozostała reszta do treningu. Sytuacja powtarza się osobno dla każdego kolejnego podzbioru. Uzyskane wyniki uśrednia się. Ma to zapobiec niezbalansowaniu zbiorów. Niezbalansowanie oznacza sytuację, w której jeden zbiór zawiera dużą większość próbek należących do tej samej kategorii.

Rys. 2.1. Walidacja krzyżowa

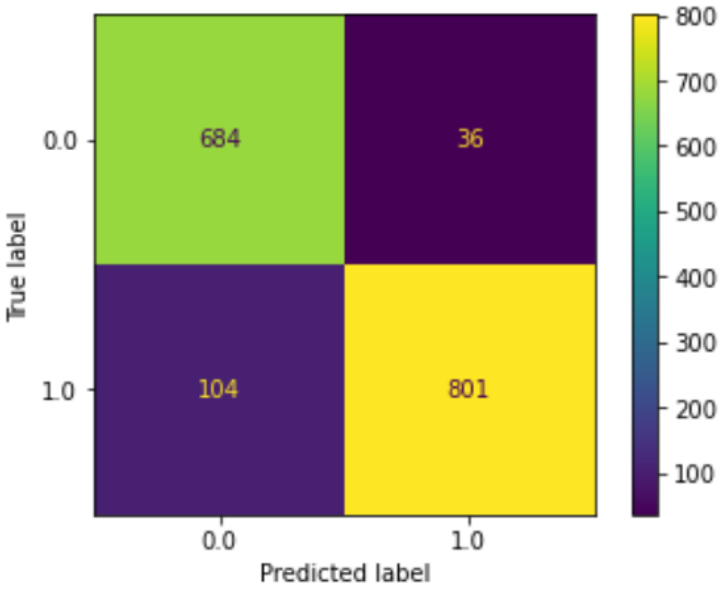
Aby oszacować model, można użyć trzech metryk:

- dokładności (ang. *accuracy*), czyli jak dużo próbek zostało poprawnie sklasyfikowanych na tle wszystkich próbek,
- czułości (ang. *recall*), czyli jak wiele próbek należących do jednej kategorii zostało poprawnie sklasyfikowanych, na tle wszystkich próbek z tej kategorii,
- precyzji (ang. *precision*), czyli jak wiele próbek, które zostały sklasyfikowane do jednej kategorii, faktycznie należało do tej kategorii.

Żeby lepiej zrozumieć różnice pomiędzy tymi metrykami, można sobie wyobrazić sytuację, w której trzeba sklasyfikować sto różnych raportów o błędach do osób, które się nimi zajmą. Raporty są z trzech różnych dziedzin: oprogramowania, sprzętu fizycznego oraz logistyki. Osoby to trzech specjalistów z poszczególnych dziedzin. Następuje przydzielenie raportów i 70 na 100 trafia w ręce odpowiedniego specjalisty. Dokładność wtedy wynosi 70%. Następnie okazuje się, że model przypisał 30 raportów do osoby zajmującej się logistyką, lecz 12 z nich należało przypisać do eksperta oprogramowania lub sprzętu. Wtedy, precyzja dla kategorii logistyka wynosi 60% (18 dzielone przez 30). Finalnie, na 36 raportów, które miały trafić do eksperta od logistyki, 18 zostało poprawnie sklasyfikowanych. Wtedy czułość kategorii dla logistyka wynosi 50% (18 dzielone przez 36)[3]. Dodatkowo, istnieje jeszcze jedna metryka używana przy szacowaniu modelu, zwana wskaźnikiem F1. Opisuje ona związek pomiędzy precyzją a czułością modelu.

W celu obliczenia tych metryk w praktyce, można posłużyć się tablicą pomyłek (inaczej macierzą błędów). Przykładowa tablica pomyłek dla klasyfikacji binarnej (czy próbka należy do kategorii lub nie) została pokazana na rys. 2.2. i składa się z czterech wartości:

- TP (ang. *True Positive*) - ilość próbek poprawnie sklasyfikowanych jako należące do kategorii (684),
- FP (ang. *False Positive*) - ilość próbek błędnie sklasyfikowanych jako należące do kategorii (36),
- TN (ang. *True Negative*) - ilość próbek poprawnie sklasyfikowanych jako nie należące do kategorii (104),
- FN (ang. *False Negative*) - ilość próbek błędnie sklasyfikowanych jako nie należące do kategorii (801).



Rys. 2.2. Macierz błędów

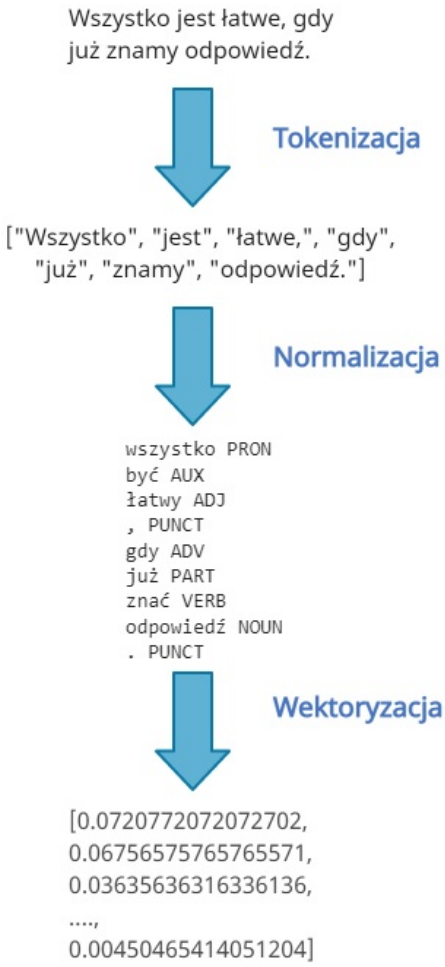
Wówczas, dysponując tymi wartościami można obliczyć dokładność, precyzję, czułość i F1 według wzorów:

- ***Dokładność*** = (TP+TN)/(TP+FP+FN+TN),
- ***Precyzja*** = TP/(TP+FP),
- ***Czułość*** = TP/(TP+FN),
- ***F1*** = 2TP/(2TP+FP+FN)[4].

W przypadku liczenia poszczególnych metryk dla większej ilości kategorii, należy zsumować odpowiednie wartości z tablicy pomyłek.

2.3. Etapy reprezentowania tekstu

W dziedzinie przetwarzania języka naturalnego (NLP) reprezentowanie tekstu jest najważniejszym procesem, umożliwiającym maszynom przetwarzanie i zrozumienie danych tekstowych. Ten podrozdział opisuje główne etapy reprezentowania tekstu: tokenizację, normalizację i wektoryzację.



Rys. 2.3. Etapy reprezentowania tekstu

2.3.1. Tokenizacja

Tokenizacja to pierwszy krok potoku NLP i dlatego ma duży wpływ na resztę potoku. Tokenizator działa na danych nieustrukturyzowanych, tekstach w języku naturalnym, tak żeby podzielić je na mniejsze kawałki informacji, które mogą zostać policzone jako dyskretne elementy. Te elementy zawierają bardziej ukierunkowane informacje[5].

Najprostszą formą tokenizacji jest dzielenie zdań na słowa po spacjach, co w Pythonie można osiągnąć za pomocą wbudowanej klasy *split*. Wykonanie tej prostej procedury pozwala na policzenie częstości występowania poszczególnych słów. Jednak, jest to metoda przydatna tylko dla języków, które posiadają wyraźne granice między słowami, jak angielski czy polski.

Niestety, użycie *splita* nie jest doskonałe z prostego powodu, a mianowicie istnienia wielotokenowych słów takich jak "Nowy Sącz" czy "Base Transceiver Station", które są jednym słowem, ale zawierają spacje. Powoduje to rozdzielanie tych słów na osobne tokeny, co jest nie do końca poprawne pod kątem rzeczywistości[5].

2.3.2. Normalizacja

Normalizacja, będąca kluczowym elementem preprocesowania tekstu jest drugim krokiem w potoku NLP. Normalizowanie korpusu wiąże się bezpośrednio z redukowaniem go, ponieważ tokeny mające podobne znaczenie będą łączone do jednej postaci. To rozwiązanie zmniejsza ilość tokenów w słowniku oraz poprawia powiązanie pomiędzy różnymi "pisowniami" tokenu[5]. Co więcej, jest dobrym sposobem na zabezpieczenie się przed nadmiernym dopasowaniem modelu do zbioru testowego, ponieważ zmniejsza liczbę słów przyjmowanych na wejściu, jednocześnie uogólniając odmiany morfologiczne tego samego słowa.

	Text	Lemma	Stem	POS	is stopword?
0	Merging	merging	merg	NOUN	False
1	fails	fail	fail	VERB	False
2	on	on	on	ADP	True
3	3	3	3	NUM	False
4	branches	branch	branch	NOUN	False
5	no	no	no	DET	True
6	implementations	implementation	implement	NOUN	False
7	will	will	will	AUX	True
8	be	be	be	AUX	True
9	carried	carry	carri	VERB	False
10	out	out	out	ADP	True

Rys. 2.4. Lematyzacja, stemming, część zdania i stop words

Do normalizacji tekstu można zaliczyć następujące metody, które zostały zaprezentowane w tabelce na rys. 2.4. przy użyciu bibliotek SpaCy i NLTK:

- ujednolicanie wielkości liter,
- usuwanie tak zwanych *stop words*, które nie zawierają istotnej informacji semantycznej,
- identyfikacja wspólnego rdzenia wśród różnych postaci słowa (*stemming*),
- wyszukiwanie formy podstawowej wyrazu (*lematyzacja*)[6],
- określanie roli słowa w zdaniu (ang. *POS tagging*)[5].

Analizując zebrane informacje *stemming* uporządkowywuje słowa w grupy, co poprawia czułość, jednocześnie zmniejszając precyzję. Dzieje się tak, ponieważ liczba wartości prawdziwie pozytywnych (TP) lub podobnych semantycznie dokumentów ulegnie zwiększeniu (co wpłynie pozytywnie na czułość), lecz równocześnie liczba wartości fałszywie pozytywnych (FP) lub dokumentów nieistotnych pod względem podobieństwa wzrośnie, co wpłynie negatywnie na precyzję. Natomiast lematyzacja może poprawić precyzję, ponieważ lemat, w odróżnieniu od rdzenia jest poprawnym słowem, formą bazową. Aczkolwiek wiąże się to z utratą informacji[5]. Z kolei wyrzucenie słów znajdujących się na liście stop words nie zawsze będzie pomocne (wyrzucenie słowa "out " zmieni znaczenie *phrasal verba* "carry out" na po prostu "carry", co jest niechcianym zachowaniem).

2.3.3. Wektoryzacja

Gdy słowa są już pogrupowane na rdzenie i lematy, należy sprawdzić ich znaczenie w tekście. Ostatni etap, czyli wektoryzacja to przekształcanie słów w liczby rzeczywiste, które mogą być używane przez modele uczenia maszynowego. Metody wektoryzacji dzielą się na:

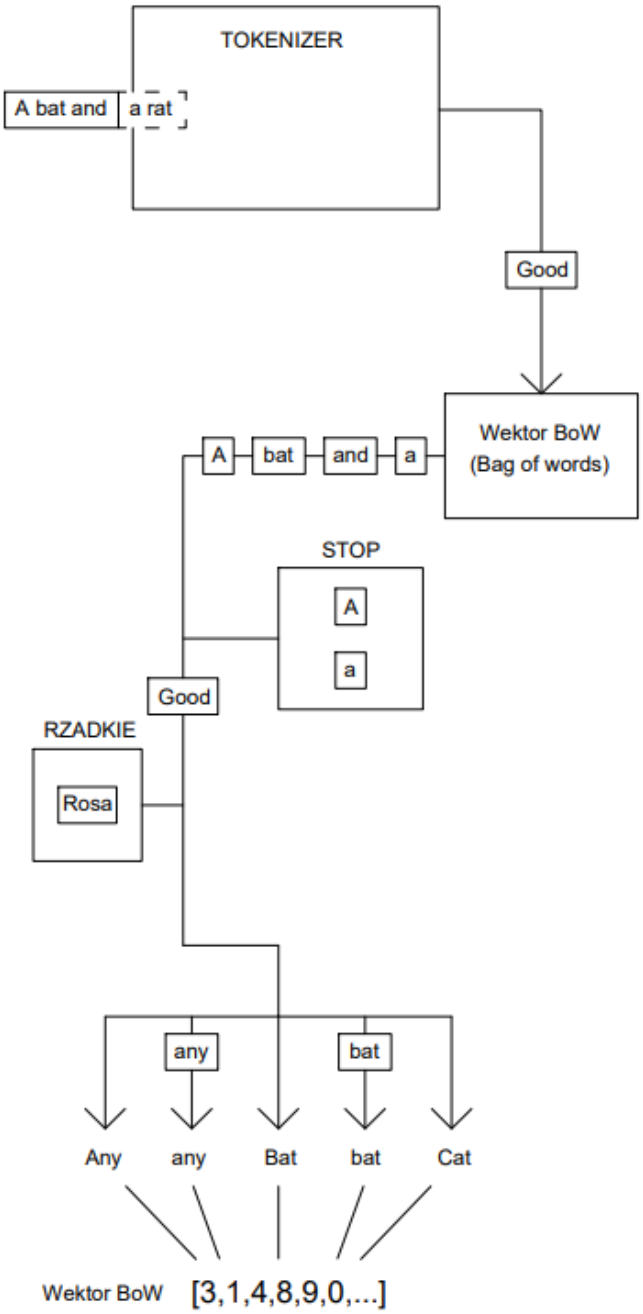
- kodowanie zero-jedynkowe (*one-hot encoding*),
- Bag-of-words - wektory liczników częstości słów,
- TF-IDF - wektory liczbowe reprezentujące również znaczenie słowa w korpusie,
- Word2Vec - model trenowany na korpusie, który jest w stanie zapamiętać informacje na temat związków między słowami i ich podobieństw.

Po udanej wektoryzacji korpusu tekstowego, można powiedzieć że słowa zostały umieszczone w n-wymiarowej przestrzeni wektorowej, gdzie n oznacza liczbę unikatowych słów w całym słowniku. Gdyby ów przestrzeń była dwu-wymiarowa, jak zwyczajna mapa, to można by policzyć odległość między dwoma punktami w tej przestrzeni porównując współrzędne X i Y, które dokładnie opisują położenie tych punktów. W przypadku przestrzeni n-wymiarowej wektorów słów, sytuacja jest analogiczna, z taką różnicą, że położenie słów jest opisane za pomocą n współrzędnych. Nie mniej jednak, da się obliczyć jak daleko są od siebie położone, co wskaże na ich podobieństwo semantyczne.

Odległość między dwoma wektorami może być mierzona na różne sposoby. Jednym z nich jest podobieństwo cosinusowe, czyli cosinus kąta pomiędzy dwoma wektorami. Podobieństwo cosinusowe to odwrotność odległości cosinusowej, a można je policzyć posługując się wzorem:

$$cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{|A||B|} \tag{2.1}$$

gdzie A i B to wektory, a cosinus ich kąta theta otrzymujemy przez podzielenie ich iloczynu skalarnego przez iloczyn ich norm[7].



Rys. 2.5. Przykładowa ilustracja Bag-of-words na podstawie [5]

Reasumując, można użyć miary odległości wektorów w przestrzeni wektorowej, aby dowiedzieć się czy słowa są do siebie semantycznie podobne. To samo tyczy się zdań, ponieważ można dokonać projekcji zbioru słów na przestrzeń wektorową, jeżeli tylko wszystkie słowa w zdaniu są obecne również w słowniku, na podstawie którego została stworzona ta przestrzeń.

Dużym problemem związanym z pracą na wielowymiarowych przestrzeniach wektorowych jest tak zwane **przekleństwo wymiarowości**, które odnosi się do sytuacji, gdy poprawne klasyfikowanie próbek, przy wykorzystaniu pełnego zbioru danych, jest trudne do wykonania, a wielkość wektora skutkuje większą złożonością klasyfikatora. Zwiększa się również prawdopodobieństwo nadmiernego dopasowania modelu oraz obniżenią zdolności uogólniających klasyfikatora. Dzieje się tak, ponieważ im więcej wymiarów, tym dalsze odległości wektorów w przestrzeni euklidesowej[8][5][9]. Innym problemem jest brak możliwości zwizualizowania wielowymiarowych danych na wykresie, ponieważ ludzki mózg będzie miał problem z przeanalizowaniem więcej niż trzech osi. W takim przypadku analiza głównych składowych (PCA)[10] zmniejsza liczbę wymiarów.

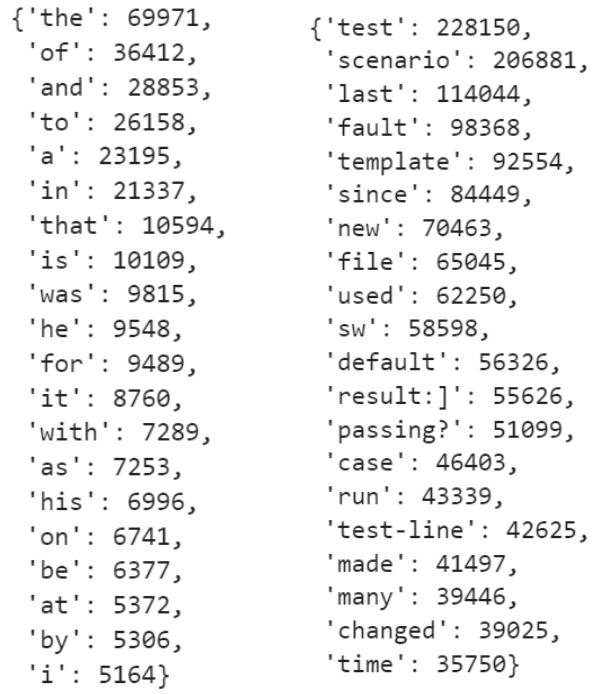
2.4. Wektoryzacja metodą TF-IDF

Jednym z najciekawszych praw funkcjonujących w dziedzinie przetwarzania języka naturalnego jest *prawo Zipfa*, które głosi że *jeżeli dla jakiegokolwiek tekstu lub grupy tekstów ustala się wykaz wyrazów ułożonych w malejącym porządku częstotliwości ich występowania, to częstość słowa jest odwrotnie proporcjonalna do numeru jego miejsca w wykazie*. Na przykład, pierwszy element w wykazie pojawi się dwa razy częściej niż drugi, a trzy razy częściej niż trzeci[5]. W obszernych korpusach językowych to prawo jest spełnione niemal doskonale dla ponad 200 pierwszych słów. Podobne zależności można zaobserwować dla częstości występowania wysokości nut w utworach muzycznych lub danych związanych z populacją w wielkich miastach[11]. Model statystyczny prawa Zipfa można wykorzystać na wiele sposobów, np. oceny jakości tekstu, oceny skuteczności algorytmów językowych lub określania, które słowa są ważne dla danego kontekstu. Ten ostatni przydaje się do klasyfikacji tekstu i wyodrębniania informacji.

Bardzo istotnym wnioskiem, jaki można wyciągnąć, obserwując rozkłady słów w korpusach tekstowych jest to, że spory procent korpusu stanowi kilka słów pojawiających się z największą częstotliwością, a większość słów występuje jedynie parę razy. Jest to ważne, ponieważ w przypadku określania przybliżonego tematu dokumentu, należy zdefiniować jego słowa kluczowe. Niekoniecznie najważniejsze słowa będą się pojawiały najczęściej. Potrzebne jest policzenie częstotliwości występowania słów w dokumencie na tle całego korpusu. Do tego można posłużyć się TF-IDF.

TD-IDF (*term frequency - inverse document frequency*) można wyrazić wzorem:

$$(tf - idf)_{i,j} = tf_{i,j} \times idf_i \tag{2.2}$$



(a) Brown corpus (b) Opisy grup

Rys. 2.6. Rozkład Zipfa dla Brown Corpus[12] oraz dla opisów grup testerskich używanych w dalszej części pracy

gdzie $tf_{i,j}$ (częstotliwość terminu) oznacza liczbę wystąpień danego słowa w danym dokumencie i obliczyć ją za pomocą poniższego wzoru:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{kj}} \tag{2.3}$$

gdzie $n_{i,j}$ jest liczbą wystąpień terminu (t_i) w dokumencie d_j, a w mianowniku jest suma liczby wystąpień wszystkich terminów w dokumencie d_j. Wielkość idf_i (odwrotna częstotliwość dokumentu) oznacza, jak rzadko dane słowo występuje w całym korpusie dokumentów, a matematycznie można to wyrazić:

$$idf_i = log\frac{|D|}{|\{d : t_i \epsilon d\}|} \tag{2.4}$$

gdzie $|D|$ to liczba dokumentów w korpusie, a $|\{d : t_i \epsilon d\}|$ to liczba dokumentów, w których termin występuje chociaż raz. Wartości TF i IDF są następnie mnożone, aby uzyskać wartość TF-IDF dla danego słowa. Słowa o wysokiej wartości TF-IDF są uważane za ważne dla danego dokumentu, ponieważ są często występujące w tym dokumencie, ale rzadko występujące w całym korpusie[13, 14]. W praktyce do mianownika dodaje się jedynkę w przypadku, gdy termin nie znajduje się w żadnym dokumencie, aby uniknąć dzielenia przez 0. Finalnie, każdemu dokumentowi zostaje przypisany wektor. Jest on zwykle wektorem rzadkim (większość elementów jest równa 0), o długości równej liczbie unikatowych słów w słowniku. Miejsca w wektorze odpowiadają poszczególnym tokenom słownika, a wektor konkretnego dokumentu jest zapełniany zerami lub współczynnikami TD-IDF, jeżeli dokument zawiera słowo. Takie

wektoryzowanie reprezentuje korpus dokumentów, jednocześnie uwzględniając znaczenie oraz unikatowość tokenu w dokumencie na tle całego korpusu [15].

2.5. Wektory tematyczne

Działania na częstości występowania terminów mogą być przydatne, ale nie powiedzą dużo o znaczeniu, jakie mają słowa. Oszacowanie znaczeń słów lub zdań w dokumencie jest trudnym zadaniem, ale w nagrodę można otrzymać dodatkowo cenne informacje i znacząco polepszyć klasyfikację. Żeby uzyskać lepszy obraz wektorów tematycznych, można wyobrazić sobie losowy odcień farby, a następnie kombinację podstawowych kolorów farb wchodzących w skład tego odcienia. Wektory tematyczne składają się z kombinacji liniowej słów. Obliczają jej znaczenie semantyczne. Kompresują rzadkie wektory słowne TF-IDF na gęste wektory reprezentujące odrębne tematy. Liczba wymiarów wektora ulega diametralnemu pomniejszeniu. Algorytmicznym sposobem na wytypowanie wektorów tematycznych jest określanie "towarzystwa" słowa, czyli liczenie współwystępowania w tym samym dokumencie. To podejście doprowadziło do opracowania kilku technik tworzenia wektorów uwzględniających kombinacje sąsiednich słów w dokumentach lub w zdaniach[5]:

- Analiza utajonych własności semantycznych (LSA)[16],
- Analiza głównych składowych (PCA)[10],
- Liniowa analiza dyskryminacyjna (LDA)[16],
- Ukryta alokacja Dirichleta (LDiA)[16],
- Obcięty rozkład na wartości osobliwe (Truncated-SVD)[17]

Różnice między tymi technikami są dosyć subtelne i poza zakresem tej pracy. Warto jednak pamiętać, że ukryta alokacja Dirichleta jest sposobem transformacji danych z użyciem twierdzenia Bayesa (prawdopodobieństwa) i jest często używana do modelowania tematów z powodu nieco lepszych wyników i przyspieszenia obliczeń[18].

Kompresja wymiarów jest równoznaczna z uogólnieniem zdolności klasyfikujących, co mocno pozytywnie wpływa na jakość wyników. Dzieje się tak, ponieważ ze wzrostem cech w danych, rośnie skomplikowanie modelu uczenia maszynowego. Tak wytrenowany model bardzo słabo radzi sobie na nowych danych, ponieważ dochodzi do nadmiernego dopasowania (przeuczenia) modelu do zbioru treningowego, co jest zupełnym przeciwieństwem oczekiwanego rezultatu. Redukcja wymiarowości modelu jest bardzo istotnym elementem całego procesu, gdyż:

- mniej nieważnych danych daje lepszą precyzję modelu,
- mniej wymiarów oznacza większą prędkość trenowania algorytmu,

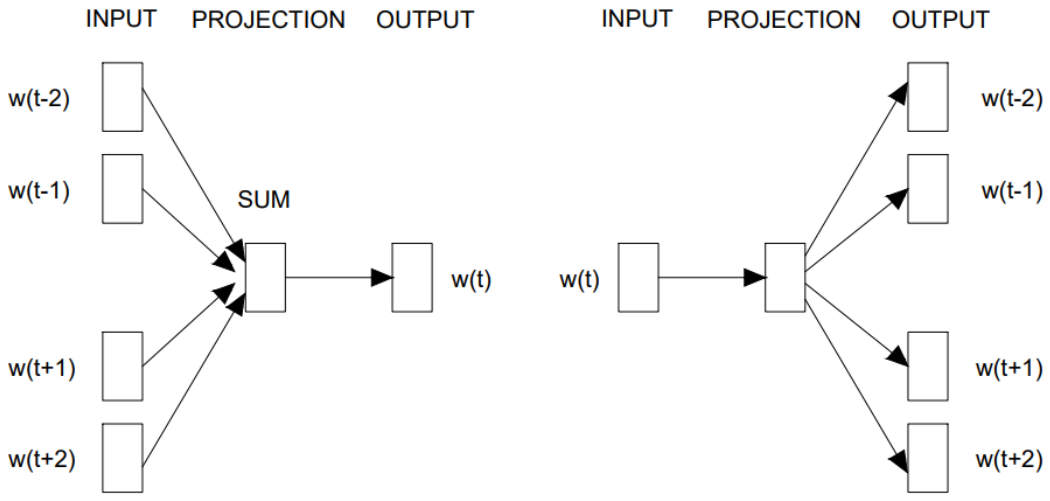
- dane zajmują mniej miejsca,
- usuwany jest zbędny szum i cechy nieskorelowane z wynikiem klasyfikacji[19].

Wiedza z tego podrozdziału stanowi broń do walki z przekleństwem wymiarowości opisanym wcześniej. Wektory tematyczne wykorzystuje się nie tylko do redukcji cech w danych, ale również do wyszukiwania semantycznego w celu znajdywania dokumentów na podstawie ich znaczenia. Klastry mogą też przydać się do wytypowania pewnych podgrup, cechujących się kluczowymi słowami, które charakteryzują obszar działania grupy testerskiej zajmującej się obsługą błędów w oprogramowaniu.

2.6. Wektory słów (Word2Vec)

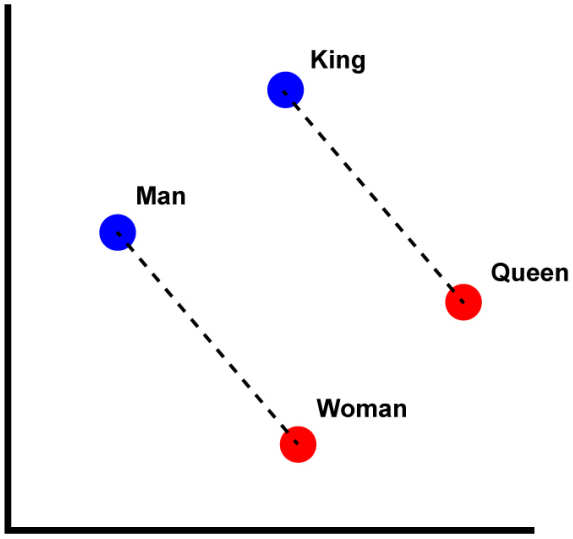
Kolejną metodą reprezentacji słow użytą w tej pracy jest, opracowany przez pracowników Google, Word2Vec. Jest to technika oparta na sieciach neuronowych, która pozwala na przekształcenie każdego słowa w wektor cech. Dzięki temu, każde słowo może być traktowane jako punkt w przestrzeni wektorowej, co umożliwia wykonywanie różnych operacji na słowach, takich jak porównywanie podobieństwa, za pomocą wbudowanych metod.

Word2Vec składa się z dwóch modeli: Continuous Bag-of-Words (CBOW) i Skip-Gram. Model CBOW działa na zasadzie przewidywania słowa docelowego na podstawie jego kontekstu, czyli słów znajdujących się w jego "sąsiedztwie". Natomiast, model Skip-gram polega na predykcji kontekstu (słów wynikowych) na podstawie danego słowa wejściowego. Oba modele są trenowane na dużym zbiorze danych, jak na przykład korpus językowy. Po treningu, każde słowo jest reprezentowane przez wektor cech. Ów wektor ma za zadanie przechwycić różne charakterystyki tego słowa na podstawie całego tekstu[5].



Rys. 2.7. Architektura modelu CBOW (po lewej) i Skip-gram (po prawej)

Model Word2Vec zawiera informacje na temat związków między słowami, włączajac w to podobieństwo. Jego efektywność bierze się z tego, że potrafi pogrupować razem wektory podobnych słów. Model ten "wie", że słowa "król"i "królowa" są bardzo do siebie podobne. Co więcej, gdyby od dwuwymiarowego wektora słowa "król" odjąć dwuwymiarowy wektor słowa "mężczyzna", a następnie dodać dwuwymiarowy wektor "kobieta", wynikiem byłby wektor bardzo zbliżony do wektora słowa "królowa"[20].



Rys. 2.8. Wektory cech

Jeszcze jedną zaletą wektorów cech wygenerowanych za pomocą Word2Vec-a jest łatwość w wizualizowaniu ich na dwuwymiarowych mapach semantycznych. Wykorzystująca PCA projekcja wektorów słów geograficznych może ujawnić kulturową bliskość miejsc[5]. W przypadku słownictwa z zakresu telekomunikacji występującym w tej pracy, mieszanka wektorów jest dużo bardziej różnorodna i o większej zawartości informacyjnej. Nie mniej jednak, związki semantyczne moga mieć wielki potencjał, a ich wizualizacje mogą prowadzić do wielu odkryć.

Sieć neuronowa, na której bazuje Word2Vec, uczy się za pomocą algorytmu propagacji wstecznej[21]. Rozszerzonym zastosowaniem jest model Doc2Vec[22], który reprezentuje całe dokumenty za pomocą wektorów. Wewnątrz korzysta z nieco innych modeli niż Word2Vec:

- Distributed Model, który przewiduje dokument, znając kontekst,
- Distributed Bag-Of-Words, predykujący słowo w dokumencie, znając dokument oraz pozycje tego słowa w dokumencie

2.7. Metody klastrowania danych

Klastrowanie danych, w przeciwieństwie do klasyfikacji i regresji, należy do nienadzorowanych technik uczenia maszynowego. Oznacza to, że algorytm samodzielnie dzieli punkty danych w grupy, nie

znając pierwotnych klas. Jest to przydatne w wykrywaniu głęboko zakodowanych ukrytych zależności w wektorach słów. Istnieje wiele różnych sprawdzonych algorytmów, na przykład:

2.7.1. Metoda K-średnich

Powszechny algorytm wśród danologów. Zastosowane kroki to:

- losowe wygenerowanie K (wartość wybrana przez programistę) centroidów, wśród punktów danych, stanowiących centrum klastra,
- obliczenie odległości euklidesowej[9] pomiędzy każdym pojedynczym punktem w danych i każdym centroidem, a następnie przydzielenie punktów do najbliższego centroidu,
- obliczenie średniego położenia dla każdego klastra i zastąpienie nim dotychczasowego centroidu, a następnie powtarzanie drugiego kroku, dopóki średnie położenia przestaną się zmieniać,
- powtórzenie poprzednich kroków N razy (wartość również wybierana przez programistę) i wybranie najlepszych klastrów na podstawie wartości zmienności (im niższa tym lepsza)[23, 24]

2.7.2. DBSCAN

DBSCAN to metoda, która kładzie nacisk na zagęszczenie punktów danych. Algorytm dzieli punkty na trzy rodzaje:

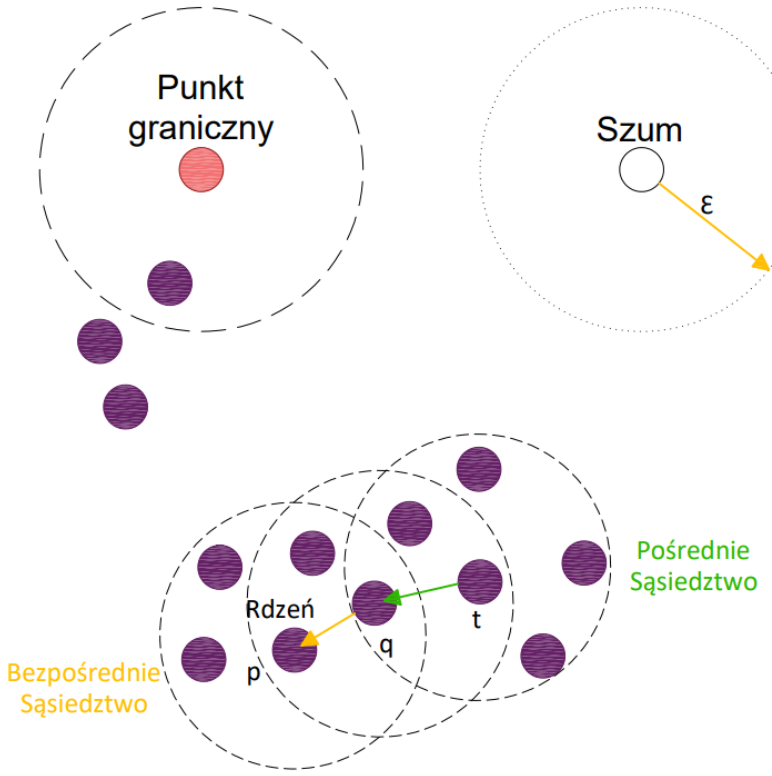
- punkty rdzenia, które w promieniu równym *eps* mają co najmniej *min_samples* sąsiadujacych punktów (obie zmienne są ustawiane manualnie),
- punkty graniczne, które nie mają wystarczającej ilości sąsiadów, żeby zostać punktem rdzenia,
- punkty szumowe, które nie mają żadnego sąsiadującego punktu

Celem tej metody jest odnalezienie obszarów o większym zagęszczeniu, oddzielonych miejscami o mniejszej gęstości. Upraszczając, algorytm będzie przydzielał sąsiadujące punkty rdzeniowe i graniczne do jednego klastra, dopóki promienie kolejnych punktów rdzenia stykają się. W przeciwnym przypadku powstanie nowy klaster[25, 26].

2.7.3. GaussianMixture

Ten model zakłada, że dane pochodzą z wybranej ilości rozkładów normalnych (*n_components*). Korzysta z algorytmu EM (Expectation-Maximization)[27], który jest statystycznym sposobem na wytypowanie parametrów tych rozkładów. Parametrami są m.in. średnia i kowariancja. Kroki działania EM to:

- E: obliczenie prawdopodobieństwa przynależności próbki do każego z rozkładów, np: "Report PR123456 na 7 procent pochodzi z rozkładu 1, na 5 procent z rozkładu 2, itp."(model uczy się wstępnych rozkładów z innego algorytmu, np. K-średnich)



Rys. 2.9. DBSCAN

- M: aktualizowanie parametrów rozkładu Gaussa z kroku E, w celu zmaksymalizowania prawdopodobieństwa przynależności próbki do rozkładu

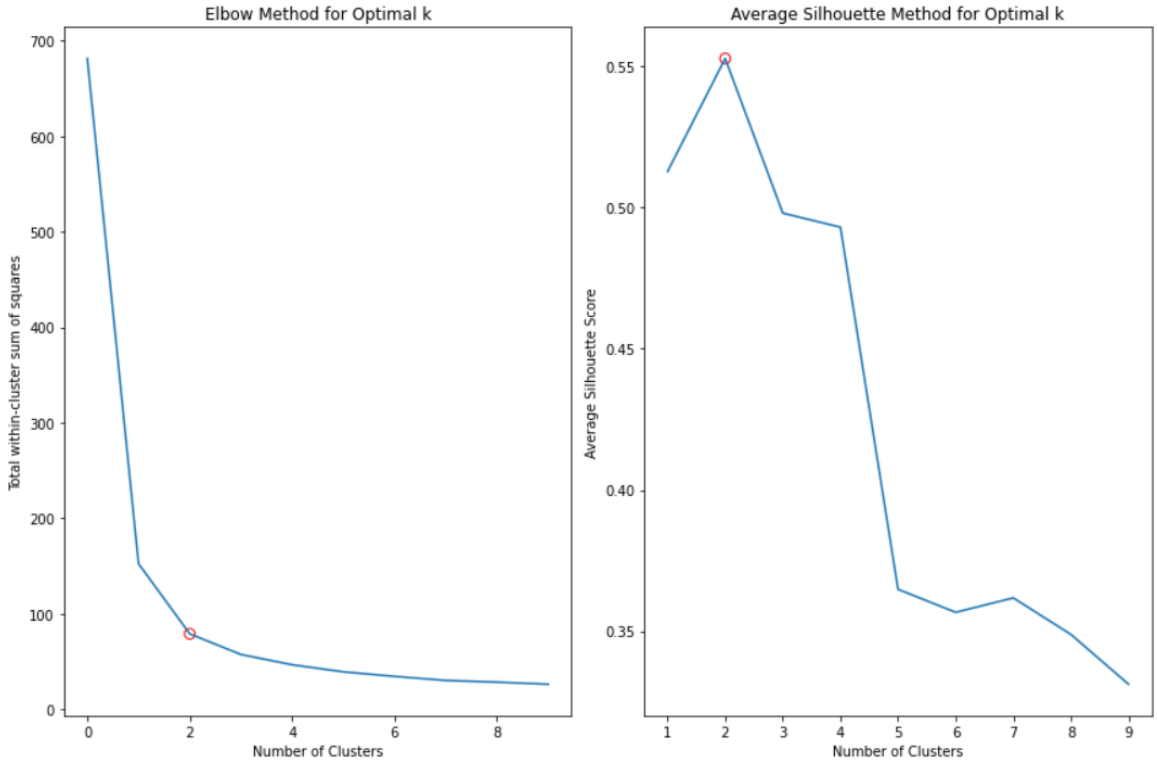
W przypadku wielowymiarowych danych, kształt rozkładu Gaussa wzdłuż każdego wymiaru jest opisany przez **macierz kowariancji**. W modelu z biblioteki sklearn[28] występują 4 rodzaje struktur kowariancji:

- *Full* - rozkłady mogą zaadaptować każdy kształt i pozycje,
- *Tied* - każdy rozkład ma ten sam kształt, ale może on być jakikolwiek,
- *Diagonal* - kształy mogą się różnić, lecz orientacja wzdłuż osi musi być taka sama,
- *Spherical* - jak diagonalna, ale kształy powinny być koliste

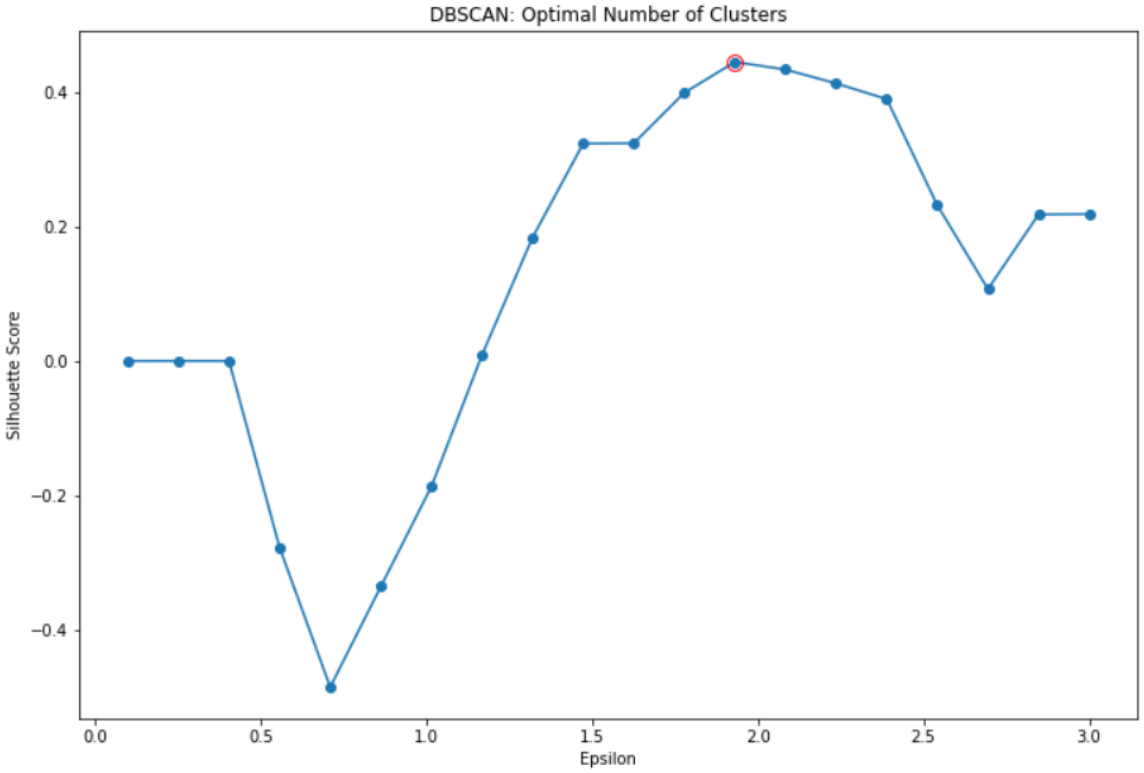
2.7.4. Dobór parametrów

Największym wyzwaniem w używaniu modeli uczenia nienadzorowanego do grupowania jest odpowiedni dobór parametrów, aby uzyskać czyste i znaczące klastry. W przypadku metody K-średnich i DBSCAN zostały użyte metody *Silhouette Score* lub sumy odległości kwadratowych[29]. Dla DBSCAN,

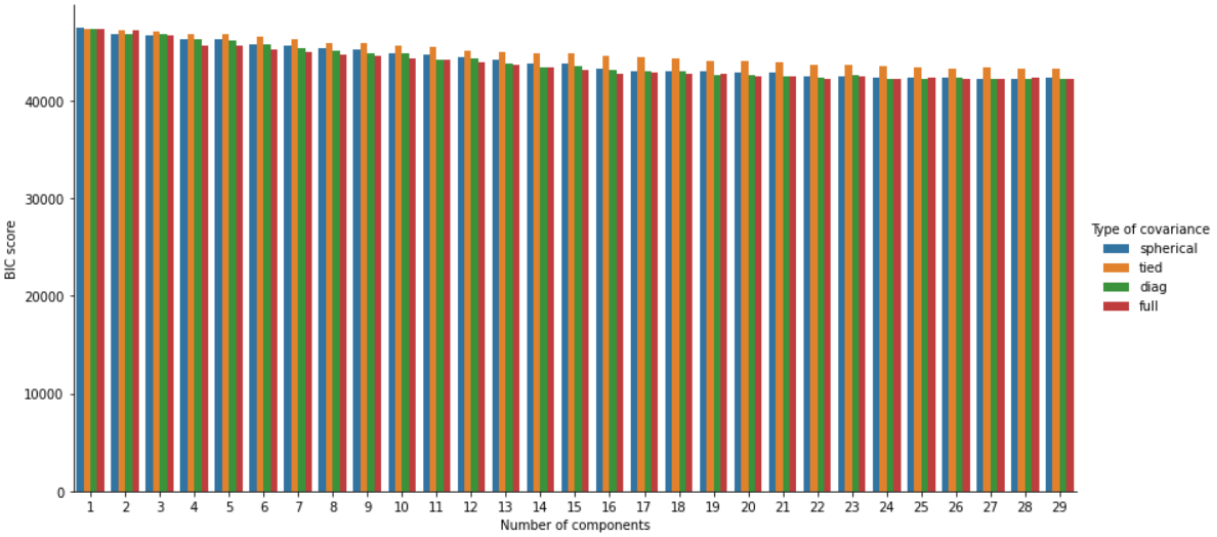
wartość *minPts* można ustawić manualnie, korzystając z zasady że powinna ona wynosić dwa razy więcej niż wymiarowość danych, a drugi istotny parametr dla tego modelu, *eps* należy wtedy odczytać z wykresu zależności ilości klastrów od średniej wartości *Silhouette Score*. Żeby wyznaczyć *n_components* oraz typ macierzy kowariancji dla modelu GaussianMixture, należy posłużyc się BIC (Bayes Information Criterion)[30].



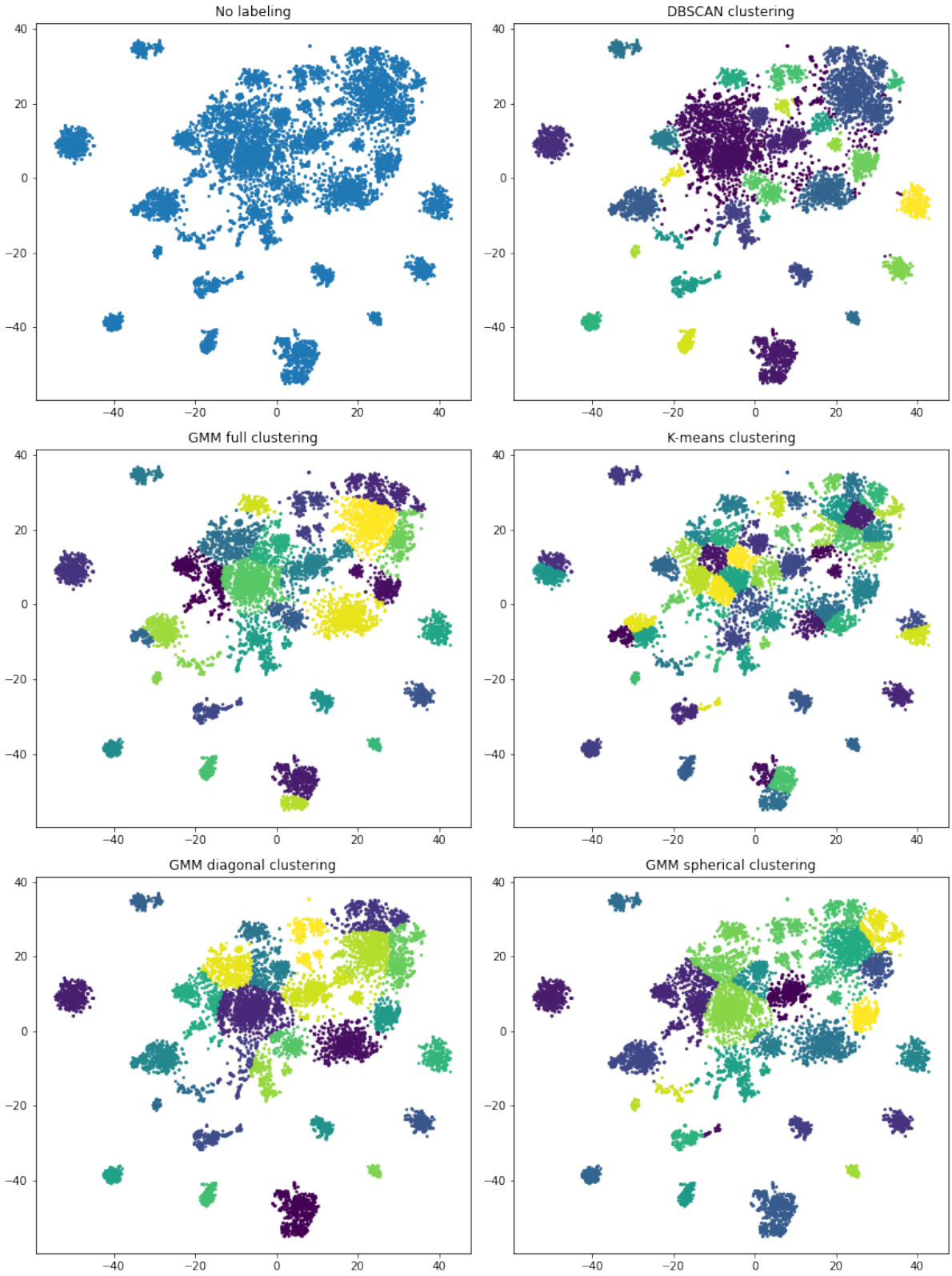
Rys. 2.10. Wyznaczenie punktu łokciowego oraz momentu nasycenia metryki Silhouette Score



Rys. 2.11. Wyznaczanie Epsilon dla DBSCAN na podstawie Silhouette Score



Rys. 2.12. Wartości BIC dla różnej ilości komponentów i rodzajów macierzy kowariancji



Rys. 2.13. Porównanie metod klastrowania

2.8. Podejścia wizualizacji danych

Dysponując wielowymiarowymi wektorami, przechowującymi cenne informacje o korpusach tekstowych można wytrenować model sztucznej inteligencji, jednakże jest to dopiero połowa sukcesu. Da się uzyskać diametralną różnicę w zrozumieniu swojego modelu i przestrzeni wektorowej, umiejętnie ją obrazując. Niestety, nie jest to takie proste, ponieważ wysokowymiarowe wektory są trudne do zobrazowania i zrozumienia przez ludzki mózg. Rozwiązaniem może być skorzystanie z metod redukcji wymiarów. Jedną z najczęściej używanych technik wizualizacji jest t-SNE (t-distributed stochastic neighbor embedding)[31], która statystycznie wydziela każdemu wektorowi miejsce na dwu lub trójwymiarowej mapie. Modelowanie odbywa się w taki sposób, że podobne wektory stają się bliskimi punktami, a dalekie punkty są wektorami niepodobnymi do siebie. Algorytm t-SNE używa rozkładu prawdopodobieństwa, aby przyporządkować rosnące wartości coraz bardziej podobnym punktom w przestrzeni wektorowej. Jest szeroko używany do zagadnień przetwarzania języka naturalnego, ponieważ dzięki tej technice można zobrazować jasno zdefiniowane klastry w zbiorze danych, co może dać lepszy wgląd na dystrybcję próbek i analizę problemu[32].

Pomimo, że t-SNE jest niezawodnym sposobem, jego użytkownik musi liczyć się z długim czasem wykonywania obliczeń i niską wydajnością. Ponadto, dobranie odpowiednich parametrów do wykresu t-SNE bywa trudnym zadaniem. Do parametrów t-SNE należą m.in:

- *perplexity*, które musi przyjąć wartość mniejszą niż ilość próbek i ma duży wpływ na uformowanie klastrów[33, 31],
- *n_components* - wymiar wektorów po redukcji[31],
- *n_iter* - maksymalna ilość iteracji w celach optymalizacyjnych[31].

Podsumowywując, t-SNE pozwala na przekształcenie wysokowymiarowych wektorów w niższym wymiarze, co ułatwia ich wizualizację i interpretację. Warto pamiętać, że przed użyciem t-SNE, powinno się zredukować wymiarowość danych inną metodą, taką jak PCA lub TruncatedSVD, w celu pozbycia się nadmiarowego szymu i przyspieszenia obliczeń[31].

3. Część praktyczna

3.1. Przedstawienie problemu

W dzisiejszych czasach sieci komórkowe stają się coraz bardziej skomplikowane i rozległe. Rośnie potrzeba na wysokiej jakości media komunikacyjne o wysokiej przepustowości i globalnym zasięgu. Spowodowane jest to zwiększającą się populacją użytkowników sieci, ale również nowatorskimi pomysłami światów wirtualnych jak na przykład metaświat. Tysiące klientów firmy wykorzystuje rozwiązania telekomunikacyjne firmy, a oprogramowanie stacji bazowych 5G jest jednym z kluczowych elementów ich funkcjonowania. Stały i nieprzerwany rozwój oprogramowań jest długotrwałym procesem, który wymaga nieustannych poprawek i ulepszeń. Przykładem może być *Test Driven Development (TDD)*, które polega na napisaniu testu konkretnej funkcjonalności w kodzie w celu walidacji, żeby następnie na tym oprzeć implementację faktycznego i zweryfikowanego działania. Są to tak zwane testy jednostkowe, które sprawdzają pojedyncze zachowania programów i stanowią one jedynie pierwszy etap testowania. Oprócz testów jednostkowych, można wyróżnić inne etapy testowania:

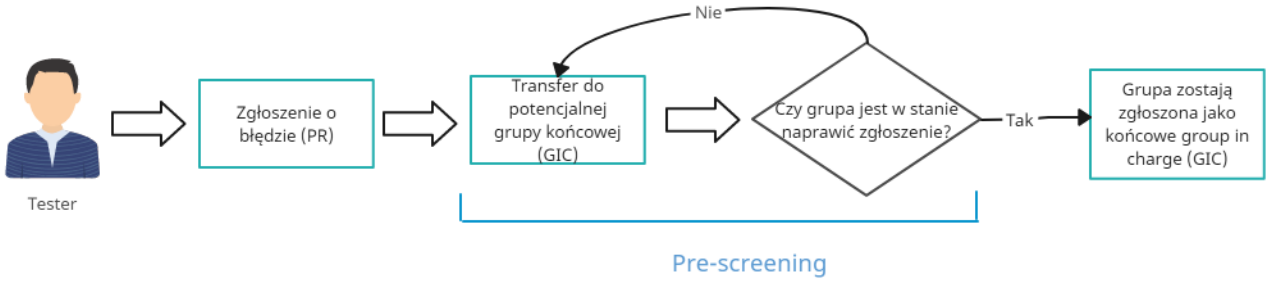
- testy integracyjne - testują wiele jednostek kodu i użytkowanie różnych systemów np. bazy danych,
- testy systemowe - testowanie działania całego systemu,
- testy akceptacyjne - sprawdzenie od strony użytkownika,
- testy regresji - sprawdzają czy wprowadzenie nowych zmian do kodu nie powoduje błędów w reszcie funkcjonalności,
- testy bezpieczeństwa - upewnienie się czy system jest odporny na ataki z zewnątrz, szczególnie pod kątem wycieku danych,
- testy wydajnościowe - testowanie wydajności i skalowalności, bardzo ważne dla niezawodności działania w przypadku wielu użytkowników

Nie licząc powyższych metodyk, obowiązkami testera jest również zapewnienie jakości pisanego kodu, który następnie jest przechowywany w oddzielnych i zabezpieczonych repozytoriach systemu kontroli wersji. Bezpieczeństwo zapewniają różnorodne techniki konteneryzacji (np. Docker) oraz emulacji systemów komputerowych (wirtualne maszyny). Są to bardzo istotne części rozwoju oprogramowania i

dlatego wydajna praca testerów wpływa mocno na rozwój firmy. Błędy zgłaszane przez testerów dotyczą różnych aspektów oprogramowania stacji bazowych, takich jak działanie, funkcjonalność, stabilność i tym podobne. Są one opisywane w formularzach zgłoszeń, które są rozpatrywane przed odpowiednie oddziały w firmie. Ogromny nakład pracy związany jest z powtarzeniem się niektórych problemów. Pomimo tego, że testerzy pozostają w kontakcie, nie mają możliwości wiedzieć, że ktoś z zupełnie innej grupy, funkcjonującej w innym kraju mógł mieć podobny problem i go rozwiązać. Efektem tego jest narastająca liczba powtarzających się zgłoszeń, co prowadzi do redundancji działań, problemów logistycznych lub zwiększonych kosztów naprawy. Technologia pozwalająca na odnalezienie odpowiedniej osoby lub chociaż kroków naprawczych do rozwiązania nowego problemu (który jest tak na prawdę stary) mogłaby znacznie usprawnić ten proces. Część implementacyjna skupia się na badaniach przeprowadzonych w celu odnalezienia jasno zdefiniowanego zestawu metod, mogących wchodzić w skład takiej technologii.

3.1.1. Opis modelu biznesowego

W Nokii, dedykowanym narzędziem, służącym do składania i obsługi takich raportów z błędami (potocznie nazywanych PR-ami od Problem Report lub Prontami) jest Reporting Portal. Jest to system udostępniający szablon według którego tester wysyła zgłoszenie. Zawiera pola ze zmiennymi kategorycznymi, tekstowymi oraz datami. Podczas wypełniania zgłoszenia, autor musi wybrać grupę prescreeningową, która weźmie na siebie odpowiedzialność i naprawi błąd, albo przekaże informacje do innej grupy, która może mieć większą ekspertyzę w danej gałęzi rozwoju. To wszystko odbywa się pomiędzy testerami i ich koordynatorami, dlatego istnienie wytyczonych „ścieżek", po których może iść nowo stworzony PR jest wysoko prawdopodobne. Opisany wyżej proces jest nazywany **procesem pre-screeningowym**. Celem tej pracy było zmniejszenie czasu i wysiłku potrzebnych do znalezienia osoby lub zespołu, zdolnych do naprawienia usterki opisanej w nowo zgłoszonym raporcie.



Rys. 3.1. Diagram przepływu pojedynczego raportu przez proces pre-screeningu

Na rysunku 3.1. został przedstawiony przepływ jednego zgłoszenia od momentu wpisania go do systemu przez testera, aż do znalezienia odpowiedniej grupy końcowej. Głównymi korzyściami czerpanymi z tego rozwiązania jest to, że błąd w oprogramowaniu nie blokuje kompletnie pracy testera, lecz zostaje przekazany do dalszego grona specjalistów. Takie podejście tworzy również dobrze opisaną historię błędów z jakimi mierzyli się programiści w firmie. Prowadzenie kompletnej dokumentacji jest istotne pod

względem uodpornienia na powszechne błędy. Pomaga też zrozumieć co poszło nie tak w poprzednich wersjach produktu oraz lepiej zaplanować kroki, uwzględniając problematyczne obszary.

3.1.2. Przebieg badania

Badania rozpoczęto od dogłębnej analizy zbiorów danych oraz dostosowania ich do problemu predykcji grupy końcowej. Pierwszym krokiem była tokenizacja i normalizacja danych. Następnie dokonano konwersji tekstu na liczby, używając metod TF-IDF oraz Doc2Vec. Ostatnim etapem badań były eksperymenty na danych przy użyciu metod redukcji wymiarowości danych i klastrowania. Na uzyskanych klastrach został wytrenowany model uczenia maszynowego lasu losowego, aby przybliżyć ich dokładność oraz sprawdzić skuteczność metod.

3.1.3. Zbiór danych

_id	object
affected_test_instances	float64
affected_test_runs	int64
author	object
author_group_dev_unit_manager	object
author_group_dev_unit_name	object
author_group_manager	object
author_group_name	object
author_group_tribe	object
author_group_tribe_group_manager	object
author_group_tribe_manager	object
author_group_tribe_manager_name	object
author_group_tribe_name	object
blocked_test_instances	int64
build	object
closed_date	object
description	object
failed_test_instances	float64
fault_occurrence_number	int64
fot	object
group_in_charge_name	object
bu_du_of_gic_of_created_pr	object
hw_unit	object
problem_type	object

product_name	object
release	object
reported_date	object
severity	object
state	object
system_release	object
test_execution_number	float64
title	object

Tabela 3.1. Pola zbioru danych historycznych PR-ów

Pierwszym zbiorem danych (nazwanym również zbiorem A), na którym odbywał się trening modeli jest zbiór historycznych PR-ów na przestrzeni ostatnich 6 miesięcy przedstawiony na Tabeli 3.1., zawierający informacje na temat autora raportu, daty zgłoszenia oraz jak wiele testów zawiodło. Oprócz tego, dane zawierają tytuł PR-a wraz z szczegółowym opisem od testera i jaki priorytet ma zgłoszenie. W pierwotnym zbiorze, pobranym za pomocą narzędzia Extract Transform Load (ETL) znajdowały się nawet raporty z wcześniejszych lat, lecz zostały odfiltrowane, ponieważ wewnątrz grup testerskich występują rotacje obowiązków, więc starsze dane mogą wprowadzić algorytm w błąd. Oprócz daty zgłoszenia, przy filtrowaniu zostały wzięte pod uwagę takie czynniki jak:

- zmienna kategoryczna *state* - czy PR został zamknięty (w innym przypadku nie ma jasno określonej grupy końcowej),
- czy grupa autora raportu nie pokrywa się nazwą z grupą końcową,
- zmienna kategoryczna *problem_type* - czy problem dotyczył oprogramowania lub hardware'u,
- oraz czy końcowa grupa należy do jednego z dwunastu wspieranych *Development Unitów (DU)*, zapisanych w zmiennej kategorycznej *bu_du_of_gic_of_created_pr*

Z racji, że ewentualne rozwiązanie wyszukiwania końcowej grupy mogło znaleźć swoje miejsce na gałęzi produkcyjnej, z której korzysta wielu klientów, autor badania postanowił zminimalizować dane wejściowe wpisywane ręcznie przez ludzi, ponieważ są one często niepełne, zawierają błędy i mogą stanowić zbędny szum dla modelu.

Niektóre cechy zostały odrzucone z powodu braku wystarczajacej ilości próbek (dużo wartości NaN), małej ważności lub faktu, że zostają dodane dopiero po zakończeniu się inwestygacji dotyczącej grupy końcowej. Inne cechy były obecne we wcześniejszych wersjach portalu do składania błędów i wobec tego nie mogły pomóc w analizowaniu nowszych zgłoszeń. Ostatecznie, postanowiono wykorzystać do treningu następujące cechy:

- *product_name* - nazwa produktu, którego dotyczy awaria,

- *system_release* - wersja systemu,
- *title* - tytuł zgłoszenia wpisany przez testera,
- *hw_unit* - konkretna jednostka hardware'u, która uległa awarii,

Ekstrakcja najważniejszych cech wyraźnie pomniejszyła zbiór, aczkolwiek wytypowała ona najcenniejsze próbki, których liczba wynosi wiele tysięcy.

Drugim zbiorem wykorzystanym w pracy (nazywanym również zbiorem B) był zbiór opisów grup końcowych udostępniony przez liderów i menedżerów tych grup. Zbiór składa się tylko z dwóch pól. Są nimi: nazwa grupy końcowej oraz odpowiadający jej opis. Liderzy opisywali swoje grupy zgodnie z przyjętym szablonem, który wymagał uwzględnienia chociaż jednej z pięciu pozycji:

- *Domains/areas in responsibility* - obszar działania grupy (za co jest odpowiedzialna),
- *HW Unit/SW specfic group* - jednostki hardware'u i software'u, którymi grupa się zajmuje,
- *HW/SW excluded from responsibility* - jednostki, którymi grupa się nie zajmuje,
- kraj i miasto, w którym funkcjonuje grupa,
- dodatkowe informacje (np. z jakimi problemami grupa sobie wcześniej poradziła)

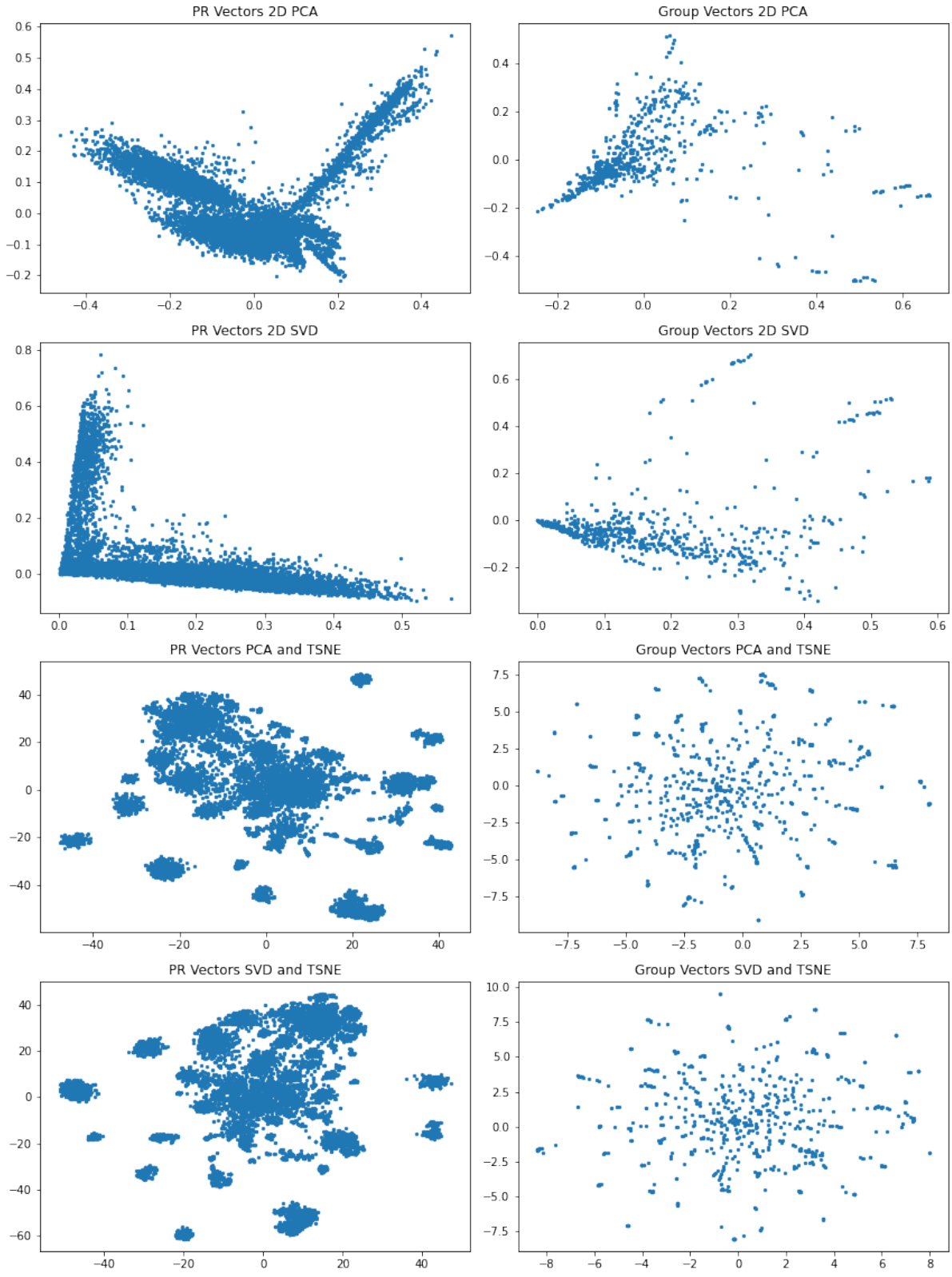
Zbiór B zawiera informacje, które posłużyły do wyszukiwania podobieństw semantycznych historycznych raportów o błędach.

Pierwszym krokiem w badaniach było sprawdzenie jak wiele grup końcowych w zbiorze B jest dobrze opisanych. Podczas analizy zbioru, można było zidentyfikować parę niechcianych przypadków, których należało się pozbyć przed przystąpieniem do predykcji. Osoby wypełniające rubryki często ignorowały szblon i używały własnych słów. Niektóre opisy świadczyły o tym, że grupa nie powinna służyć jako grupa końcowa z wielu powodów takich jak: posiadanie grupy nadrzędnej przyjmącej zgłoszenia, bycie nieaktywną grupą lub po prostu brak specjalizacji w testowaniu oprogramowania. Wszystkie takie przypadki zostały przefiltrowane za pomocą wyrażeń regularnych[34] i ostatecznie wyłoniono ze zbioru 761 dobrze opisanych grup. Autor pracy upewnił się, by część wspólna grup ze zbioru A oraz grup ze zbioru B wynosiła 1.

Na rys. 3.2 przedstawiono wektory TF-IDF obu zbiorów. W tym przypadku redukcję do dwóch wymiarów wykonano za pomocą PCA i SVD. Użyto też TSNE z wartościami *perplexity* równe 200 i *n_iter* równe 1000. Właściwości TSNE pomagają zauważyć klastry punktów danych w przestrzeni wektorowej słów.

3.2. Opis obróbki danych tekstowych

W tym celu wykorzystano potok wstępnego przetwarzania danych, którego funkcjonalność polegała na normalizacji korpusu słów oraz przygotowaniu ich w odpowiedniej formie, umożliwiającej trening modelu:



Rys. 3.2. Zbiory danych A (po prawej) i B (po lewej) na osi XY

Listing 3.1. Potok wstępnego przetwarzania danych

```
def clean_tokenize(doc):

    extended_text = replace_non_latin_chars(doc)
    extended_text = spacing_punctuation(extended_text)
    extended_text = clean_special_punctuations(extended_text)
    extended_text = remove_spaces(extended_text)
    extended_text = handle_camel_case(extended_text)
    extended_text = extended_text.split()

    for word in extended_text:

        replacement = norms.get(word)
        definition = definitions.get(word)
        extension = extensions.get(word)

        if definition and not definition['norm'] == word:
            replacement = definition['definition']

        if extension and not extension['norm'] == word:
            replacement = extension['extension']

        if replacement and not replacement == word:
            word_idx = extended_text.index(word)
            replacement = replacement.split()
            extended_text[word_idx:word_idx] = replacement

            extended_text.remove(word)

            break

    extended_text = remove_special_characters(extended_text)
    extended_text = reduce_lengthening(extended_text)
    extended_text = remove_stopwords(extended_text)

    extended_text = handle_short_words(extended_text)
    extended_text = handle_numeric(extended_text)

    extended_text = find_words_lemmas(extended_text)
    return extended_text
```

Razem z technikami normalizacji opisanych szerzej w części teoretycznej, w potoku zostało uwzględnione:

- obsługiwanie camel case'ów (np. nazwaGrupy),
- usuwanie/podmiana znaków specjalnych pochodzących z innych języków,

- usuwanie literówek (np. "ggrupa" na "grupa"),

- rozwijanie i normalizacja skrótów myślowych stosowanych przez testerów wewnątrz firmy

Generalnie, potok skupiał się na jak dokładniejszym wyczyszczeniu literówek w opisach błędów składanych przez testerów. Niektóre metody, takie jak usuwanie cyfr zostały świadomie zakomentowane, ponieważ zdarza się, że grupy mają je w nazwie.

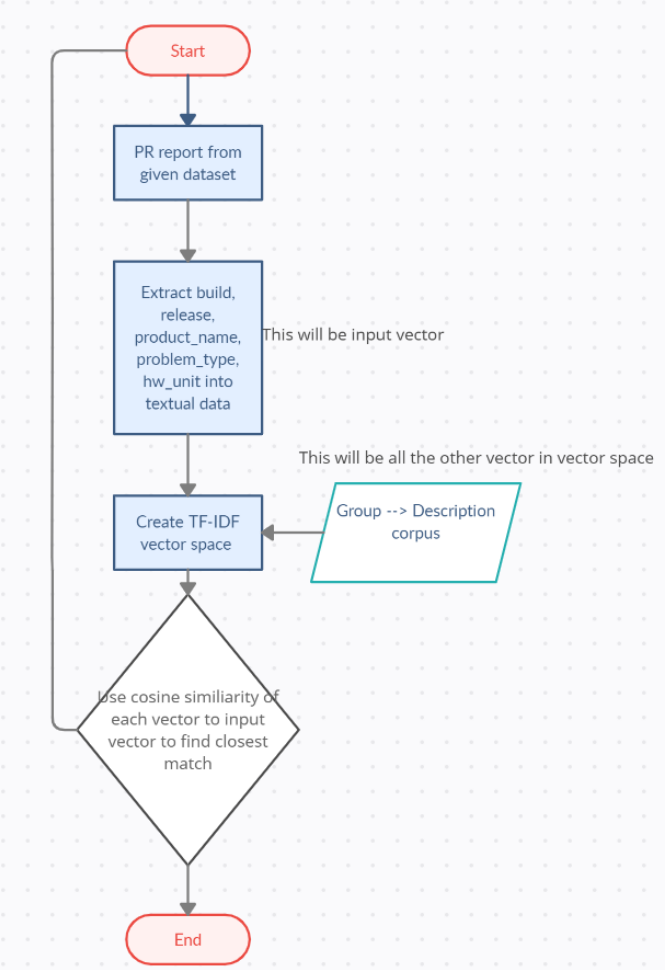
Autor zakłada, że podanie jak największej ilości dokumentów do Doc2Vec'a[22], zwiększy dokładność modelu i wzbogaci korpus językowy, co poskutkuje stworzeniem większego słownika. Dodatkowo, słowa zostały sprowadzone do "obciętych"postaci i będą się często ze sobą pokrywały co jeszcze bardziej ulepszy proces wektoryzacji. Dokumenty zostały zebrane jako lista i podane na wejście modelu na podstawie [35]:

Listing 3.2. Trening modelu Doc2Vec na korpusie dokumentów

```
1 documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(token_list)]
2 model = Doc2Vec(documents, vector_size=2, window=3, min_count=1, workers=4)
```

3.3. Podejście heurystyczne

Manualna analiza zbioru B wykazała, że duży procent zawartości tekstu stanowią słowa kluczowe, czyli nazwy własne, obszary i żargon korporacyjny. Dlatego na samym początku zdecydowano się na wektoryzację metodą TF-IDF, która wyłapywałaby słowa charakterystyczne dla danej grupy. To naiwne podejście przedstawiono na rys. 3.4. i polegało ono na iterowaniu po zbiorze A. W każdej iteracji następowała ekstrakcja wytypowanych wcześniej istotnych cech PR-a do tekstu, tak by następnie utworzyć przestrzeń wektorową wraz opisami grup. Najpierw jednak dokumenty należące do korpusu A oraz B znormalizowano przy użyciu potoku z listingu 3.2. Następnie program obliczał podobieństwa cosinusowe wektora PR-a z wektorami wszystkich grup po kolei. Algorytm celował w przewidzenie zmiennej kategorycznej *group_in_charge_name*. Ostatnim etapem było posortowanie wyników i podanie na wyjście dziesięciu największych wraz z odpowiednią nazwą grupy. Z racji na złożoność tego algorytmu, pojedyncza iteracja zajmowała dosyć długo, a wykonywanie całego programu zajęło około trzech godzin.



Rys. 3.3. Diagram przybliżający koncept pętli cosinusowej

Listing 3.3. Pętla obliczania podobieństwa cosinusowego wektora PR-a do wektorów opisów grup końcowych

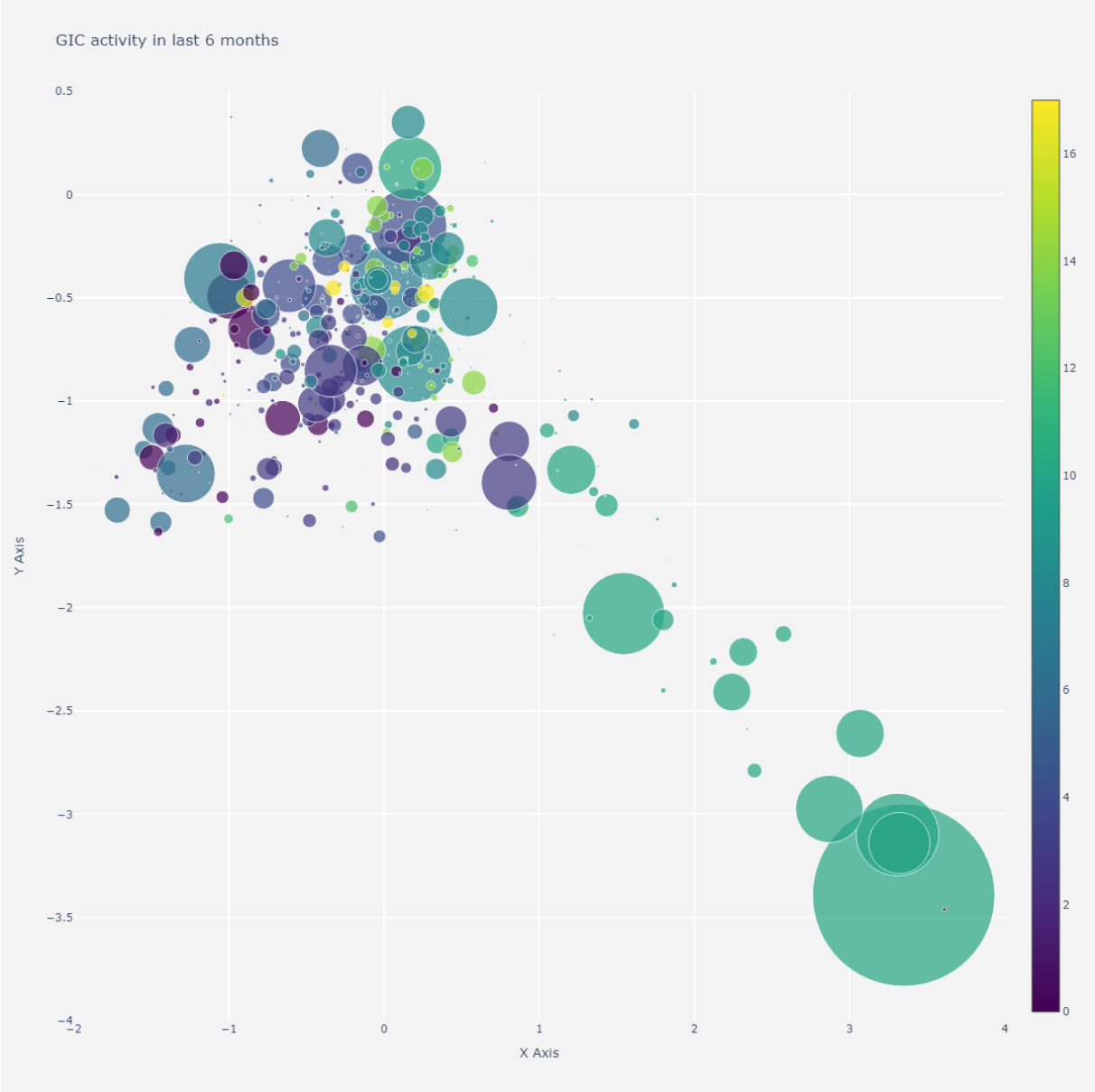
```
all_doc = []

for i,pr in pr_dataset.iterrows():

    doc = []
    doc.extend(pr['build'].split(" "))
    doc.extend(pr['release'].split(" "))
    doc.extend(pr['product_name'].split(" "))
    doc.extend(pr['problem_type'].split(" "))
    doc.extend(pr['hw_unit'].split(" "))
    all_doc.append(" ".join(doc))

vectorizer = TfidfVectorizer()
model = vectorizer.fit_transform(all_doc) #vector space

corpus = descriptions.copy() #grab all good descriptions
```

```
groups = vectorizer.transform(corpus)

pr_embeddings = list(model.todense())
group_embeddings = list(groups.todense())

for i, doc_emb in enumerate(pr_embeddings):

   groups_similiarity = []
   dataset = [*group_embeddings, doc_emb]

   for j in range(len(dataset)-1):
      groups_similiarity.append(cosine_sim(dataset[j].getA1(), dataset[-1].getA1()))

   index = groups_similiarity.index(max(groups_similiarity))
   ind = pd.Series(groups_similiarity).nlargest(10).index.values.tolist()

   print(f"For {pr_dataset.iloc[i]['_id']} top 10 are:\n
   {df5.iloc[ind]['Group Name']}\n
   real answer is: {pr_dataset.iloc[i]['group_in_charge_name']}\n")
```

Na powyższym listingu została przedstawiona dokładna implementacja w Pythonie. Badania powtórzono dla wektoryzacji metodą Word2Vec, a precyzyjniej Doc2Vec[22]. Końcowa dokładność w klasyfikacji grupy końcowej była zauważalnie niska, co spowodowało zaniechanie dalszych eksperymentów w tym kierunku.

3.4. Wyszukiwanie grup o podobnych specjalizacjach

Na wykresie bąbelkowym przedstawiono grupy ze zbioru B. Położenie na osiach X i Y jest związane z semantycznym podobieństwem ich opisów zbadanym metodą Doc2Vec. Rozmiar oraz kolory oznaczają odpowiednio ilość przyjętych zgłoszeń oraz w czym grupa się specjalizuje.



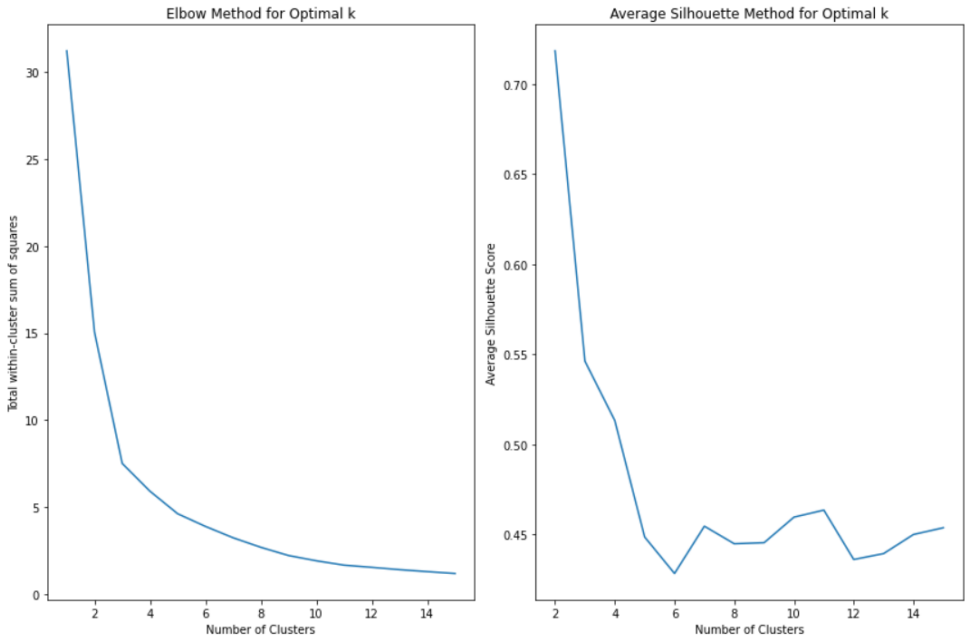
Rys. 3.4. Wykres bąbelkowy

Wykorzystanie nienadzorowanego podejścia grup testerskich w kontekście klasyfikacji oznacza, że model nie jest trenowany bezpośrednio na etykietach klas, ale na grupach lub podzbiorach danych. Istotą tego podejścia jest to, że modele mają większą swobodę wewnętrznego uczenia się struktury danych i wzorców występujących w nich.

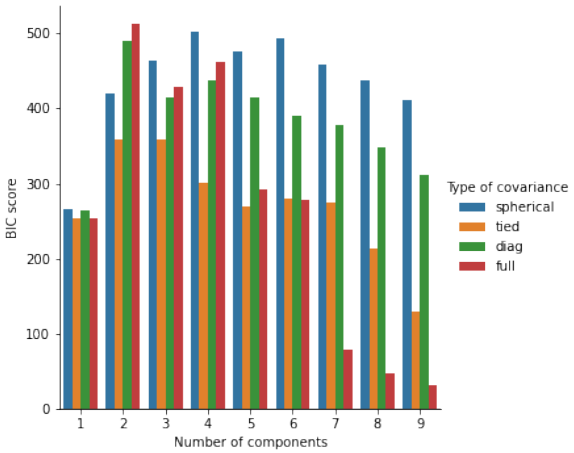
W przypadku, gdy mamy dwa zbiory danych: zbiór A, który zawiera wektory, które chcemy sklasyfikować, i zbiór B, który zawiera dane do grupowania, możemy najpierw użyć klasteryzacji na zbiorze B. Celem grupowania jest znalezienie naturalnych struktur w danych i utworzenie grup, które mają cechy wspólne.

Następnie, posiadając utworzone grupy w zbiorze B, można użyć tych informacji do ograniczenia liczby klas, do których zostaną sklasyfikowane wektory ze zbioru A. Na przykład, jeśli zbiór B został pogrupowany w 10 grup, możemy przypisać każdej grupie unikalną klasę, na przykład od 0 do 9.

Można użyć tego przypisania jako informacji wstępnej do klasyfikacji wektorów ze zbioru A, a następnie wytrenować model nadzorowany do klasyfikacji na podstawie informacji o grupach w zbiorze B.



Rys. 3.5. Punktu łokciowy oraz moment nasycenia metryki Silhouette Score dla zbioru B

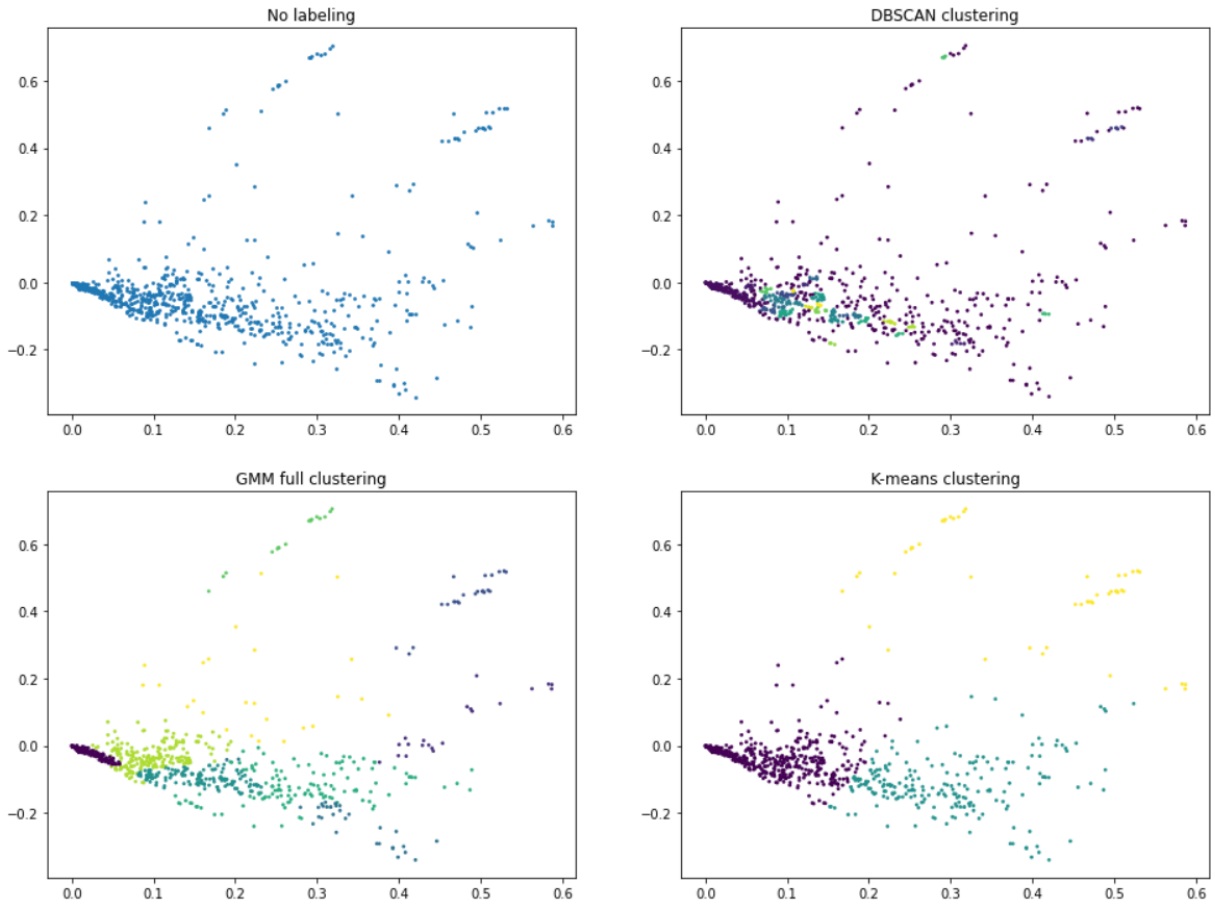


Rys. 3.6. Wyznaczanie ilości komponentów i rodzaju macierzy kowariancji (wskaźnik BIC dla zbioru B)

W tym celu dokonano parametryzacji modeli klastrowania, wykorzystując techniki i metryki omówione w części teoretycznej. W tym celu użyto wektorów TF-IDF, które zostały przekształcone za pomocą SVD. Dla algorytmów K-średnich i GMM przedstawiono odpowiednie wykresy, prezentujące wyniki klastrowania.

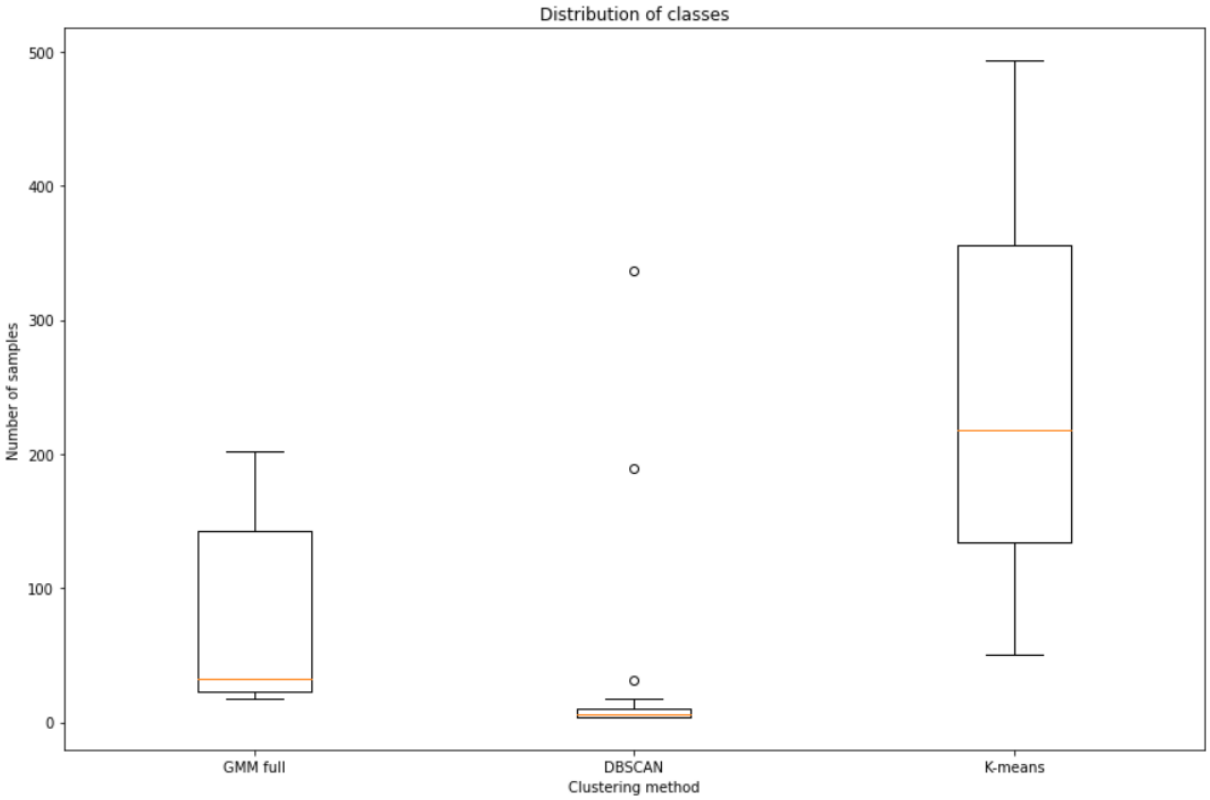
W przypadku DBSCAN przeprowadzono analizę parametrów, która wykazała, że wartość parametru *eps* powinna być zbliżona do zera. Po uwzględnieniu tych wyników, ostatecznie zdecydowano się użyć następujących modeli do klastrowania zbioru A, wraz z ich odpowiednimi parametrami.

- K-średnich (n=3),
- DBSCAN (eps=0.0067, minPts=4),
- GMM (n_components = 9, typ macierzy kowariancji "Full")



Rys. 3.7. Porównanie metod klastrowania

Dystrybucję próbek w poszczególnych klasach, w zależności od zastosowanej metody przedstawia wykres pudełkowy. Najlepsze wyniki daje definitywnie GaussianMixtureModel.



Rys. 3.8. Wykres pudełkowy przedstawiający rozkład próbek w różnych klastrach. Na przykład, dla algorytmu DBSCAN można zauważyć dużą liczbę wartości odstających, ponieważ algorytm grupuje gęste próbki w duże klastry

3.5. Budowanie predykcji na klastrach

W końcowej fazie badania wykorzystano wcześniej uzyskane klastry na zbiorze B do grupowania raportów w zbiorze A. W ramach każdego klastra przetestowano model uczenia maszynowego oparty na losowym lesie wspieranym przez BaggingClassifier z parametrem n_estimators=10 [36]. Model ten został trenowany przy użyciu 10-krotnej walidacji krzyżowej, z zastosowaniem modelu StratifiedKFold [37]. Wyniki zostały zobrazowane w podrozdziale Dyskusja.

3.6. Dyskusja

Zadanie polegające na wyszukiwaniu grupy końcowej odpowiedzialnej za naprawienie błędu zaraportowanego przez testera jest niezwykle trudne i osiągnięcie perfekcyjnej dokładności w rozwiązaniu tego zadania jest niemożliwe. Jednak jest to wyjątkowo złożony problem, którego rozwiązanie może przynieść ogromne korzyści. Automatyzacja tego procesu rozwojowego może przynieść oszczędność zarówno czasu, jak i pieniędzy. Co więcej, otwiera ona możliwość zgłębienia języka korporacyjnego, który funkcjonuje w zamkniętym społeczeństwie, skoncentrowanym na określonych celach, takim jak

duża firma. Dlatego problem ten jest na tyle interesujący, że warto przeprowadzać w jego sprawie dogłębne badania.

Złożenie formularza o błędzie kosztuje testera nie mało czasu i jest dosyć wymagającą czynnością. Wyzwanie z jakim musi sobie poradzić sprawia że staje się podatny na popełnienie błędu i wpisanie złych wartości, co pogarsza jakość predykcji. Ponadto, wielu z nich nie zawsze do końca rozumie usterkę i pomimo, że starają się jak najlepiej opisać ją własnymi słowami to nie jest to do końca usystematyzowany sposób. Gdyby opisy były idealne, a słowa czy skróty zawsze obecne w słowniku to sprawa byłaby dużo prostsza. Teoretycznie, rolą formularza w portalu do składania błędów jest uproszczenie tych niedogodności. Niestety, pola które faktycznie mają znaczenie, czyli opis usterki oraz w jaki sposób doszło się do rozwiązania są zaniedbywane i źle udokumentowane. Testerzy często nie chcą tracić czasu na opisanie dokładnych kroków podjętych przy naprawie. Również bariera językowa między pracownikami zdaje się mieć spory wpływ na zróżnicowanie raportów o błędach. W dużych firmach pracują ludzie z całego świata, a obywatel każdego państwa może mieć zupełnie inne podejście, nawet jeśli należy do tej samej grupy testerskiej.

Nie mniej jednak, przeprowadzono eksperyment mający na celu przewidzenie ostatecznej grupy końcowej za pomocą modelu BaggingClassifier, gdzie bazowym modelem był RandomForestClassifier z biblioteki sklearn. W ramach eksperymentu mierzono metryki (dokładność, precyzję, czułość i F1[38, 39]) w dwóch kategoriach: top1 i top5. Oznacza to, że sprawdzano, czy prawdziwa grupa, w której zostało zakończone zgłoszenie, znajdowała się na pierwszym miejscu lub na pięciu pierwszych miejsach w wynikach wygenerowanych przez model. Do wektoryzacji danych testowano metodę TF-IDF, rozszerzoną o redukcję wymiarowości techniką SVD.

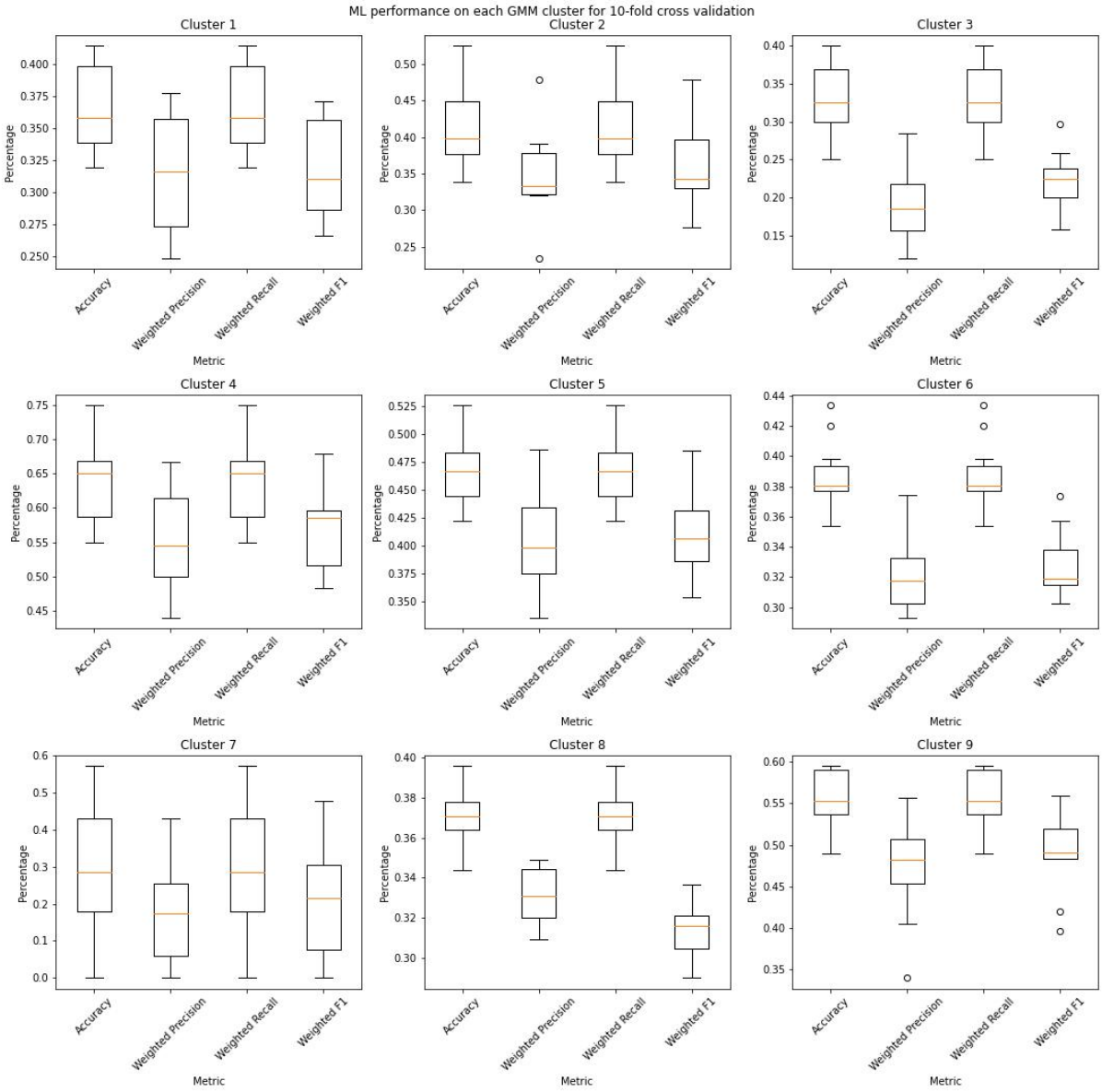
Wyniki dotyczące klastrów uzyskanych za pomocą metod K-średnich i GMM zostały zaprezentowane na wykresach pudełkowych widocznych na rysunkach 3.9 - 3.12. Postanowiono pominąć wyniki dla DBSCAN ze względu na dużą liczbę próbek przypisanych do kategorii szumu. Najwyższa dokładność procentowa została osiągnięta dla klastrów utworzonych za pomocą metody GaussianMixtureModel w kategorii top5, jak przedstawiono na rysunku 3.10. W tym przypadku analizowane metryki oscylowały w zakresie od 60 do ponad 90 procent, co oznacza nawet trzykrotne polepszenie w porównaniu z obecnymi rozwiązaniami produkcyjnymi. Metoda K-średnich, zaprezentowana na rysunku 3.12, osiągnęła nieco niższe wartości metryk, ale również wygenerowała trzy razy mniej klastrów, co powoduje większą niepewność w wynikach. Głównym powodem, dla którego K-średnich nie jest optymalnym wyborem w tym konkretnym przypadku, jest to, że metoda ta działa najlepiej dla klastrów o kształtach kulistych i wypukłych, charakteryzujących się prostą strukturą. Ponadto, warto zauważyć, że metody K-średnich i DBSCAN są w stanie wykryć tylko klastry, które sąsiadują ze sobą i oddzielone są granicami. W przeciwieństwie do tego, statystyczne podejście EM, zastosowane w implementacji GaussianMixtureModel, ma zaletę wykrywania klastrów, które się nakładają. W kategorii top1, wyniki dla obu metod zostały przedstawione na rysunkach 3.9 oraz 3.11, a także tutaj klastrowanie GMM okazało się skuteczniejsze. Ponadto, podział próbek na większą liczbę klastrów umożliwia lepsze przybliżenie obszaru, z którego potencjalnie pochodzi docelowa grupa. Na przykład, choć grupa wybrana przez algorytm może

nie być zgodna z etykietami, to w rzeczywistości może należeć do ekosystemu grup, w którym obecna jest właściwa grupa. Taki wybór zmniejsza czas potrzebny na przekazanie informacji o błędzie odpowiednim osobom.

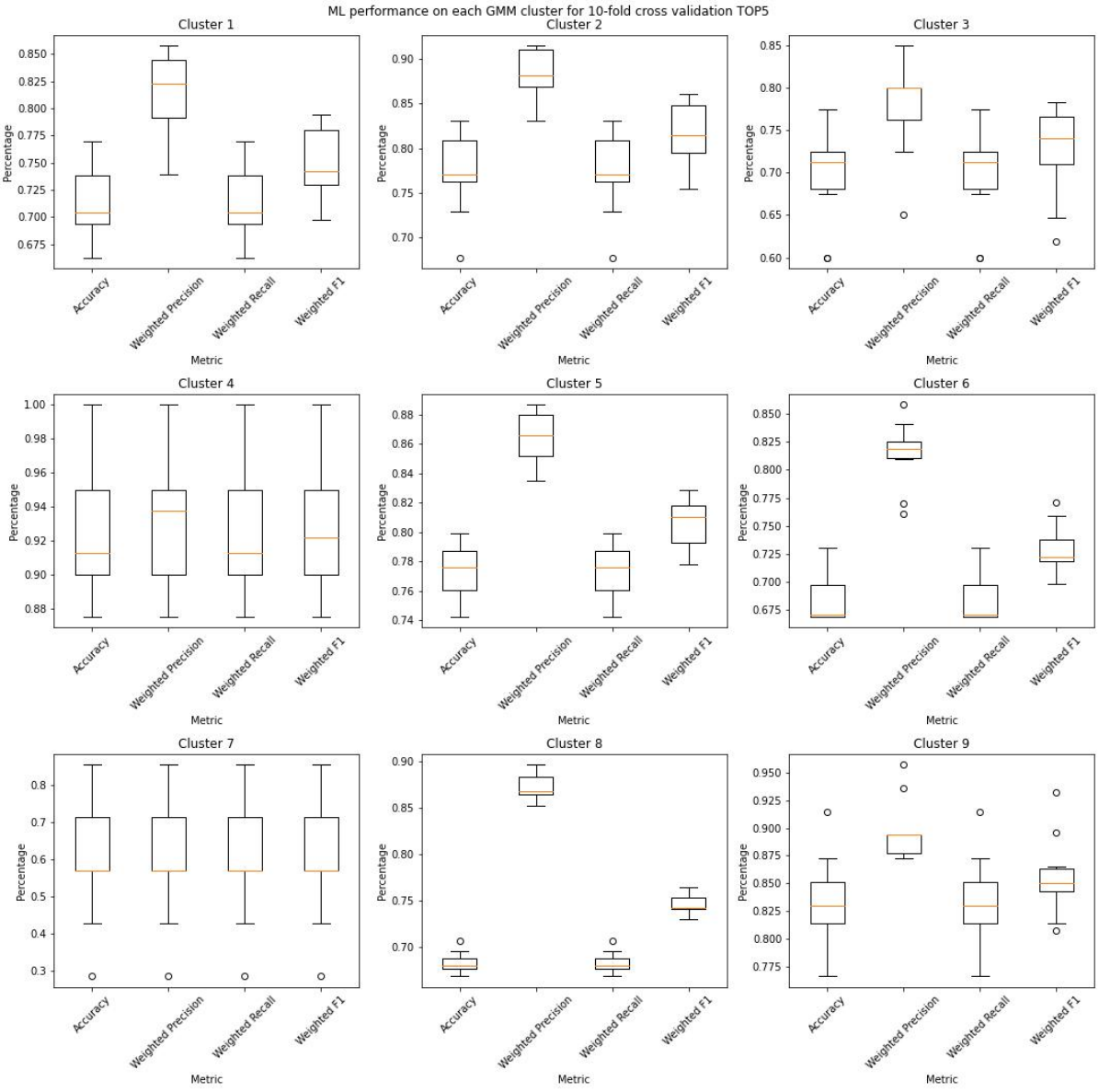
Analizując wykresy, można zauważyć, że wyniki top1 mogłyby ulec pewnemu ulepszeniu, jednakże warto zaznaczyć, że metoda top5 daje całkiem zadowalające wyniki. Wskazuje to na umiejętność modelu w identyfikacji odpowiednich grup, choć nie zawsze na pierwszym miejscu.

Należy również wziąć pod uwagę, że automatyzacja tego procesu ma duże znaczenie dla oszczędności czasu i zasobów, a uzyskane wyniki są obiecujące, szczególnie jeśli weźmie się pod uwagę metrykę top5.

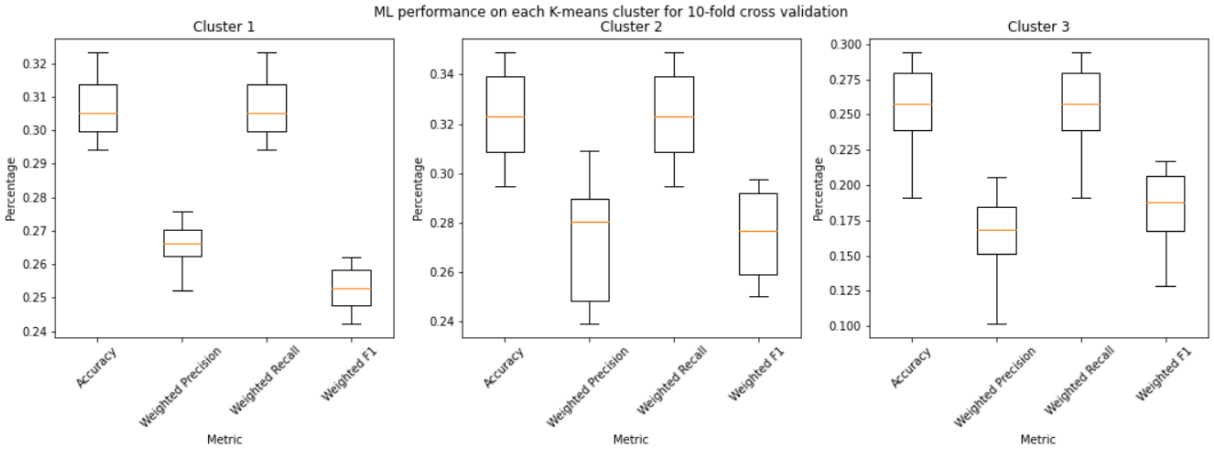
Kolejnym krokiem badania mogłoby być przetestowanie opracowanych narzędzi na prawdziwych próbkach, które nie posiadają jeszcze końcowych grup, aby dało się ustalić czy wyniki dają jakąś realną wartość. Krokiem, który wymaga również upływu czasu jest doskonalenie szablonu opisującego grupy, tak aby informacje z niej ekstrahowane były coraz bardziej użyteczne.



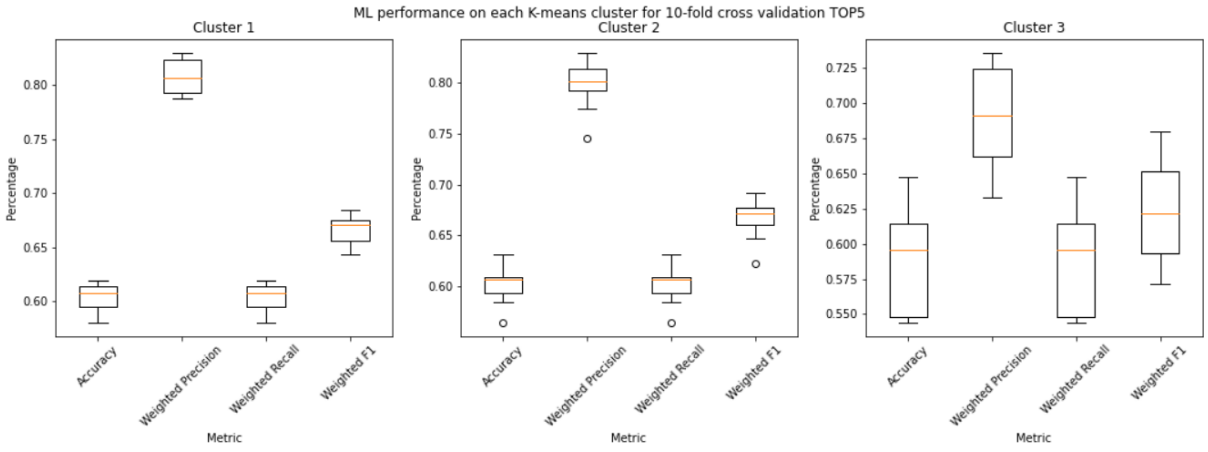
Rys. 3.9. Wyniki dla klastrów uzyskanych za pomocą metody GaussianMixtureModel (TOP1)



Rys. 3.10. Wyniki dla klastrów uzyskanych za pomocą metody GaussianMixtureModel (TOP5)



Rys. 3.11. Wyniki dla klastrów uzyskanych za pomocą metody K-średnich (TOP1)



Rys. 3.12. Wyniki dla klastrów uzyskanych za pomocą metody K-średnich (TOP5)

4. Wnioski

Głównym celem pracy było opracowanie narzędzia zdatnego do wyszukiwania grupy końcowej, która zajmie się błędem zgłoszonym przez testera w formularzu przesłanym przez portal przy użyciu technik uczenia maszynowego. Przenalizowano potok przetwarzania języka naturalnego i opisano jego najważniejsze etapy. Porównano metody wektoryzacji TF-IDF oraz Doc2Vec. Użyto metod redukcji wymiarowości takich jak, analiza głównych składowych (PCA) oraz rozkład na wartości osobliwe (Truncated SVD). Do klastrowania danych posłużyły metody DBSCAN, K-średnich oraz GaussianMixtureModel. Do wizualizacji przetestowano algorytm t-SNE. Pozyskane wyniki świadczą o tym, że starsze techniki, niekoniecznie oparte na sieciach neuronowych spisują się równie dobrze jak te nowsze. Praca podkreśla ważność dostosowywania metodyk do problemu klasyfikacji i posiadanego zbioru danych. Przeprowadzone badania stanowią gruntowny start dla dalszego rozwijania technologii opartych na uczeniu maszynowym wewnątrz firmy oraz dają szanse na wprowadzenie rozwiązania predykcji grupy końcowej na produkcję.

Bibliografia

[1] *Ciągły rozkład prawdopodobieństwa - Wikipedia, Wolna Encyklopedia*. https://pl.wikipedia.org/wiki/Ciągły_rozkład_prawdopodobieństwa Data dostępu: 2023-01-17.

[2] *Zmienne ilościowe i jakościowe*. https://www.naukowiec.org/wiedza/metodologia/zmienne-ilosciowe-i-jakosciowe_3506.html Data dostępu: 2023-01-17.

[3] *Guide to accuracy, precision and recall*. https://www.mage.ai/blog/definitive-guide-to-accuracy-precision-recall-for-product-developers Data dostępu: 2023-01-27.

[4] *Accuracy, Recall, Precision, F-Score, which to optimize on? - Towards Data Science*. https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124 Data dostępu: 2023-01-27.

[5] Hannes Max Hapke Hobson Lane Cole Howard. *Przetwarzanie języka naturalnego*. Poland: PWN, 2021.

[6] *Stemming and lemmatization*. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html Data dostępu: 2023-01-25.

[7] *Understanding cosine similiarity - Towards Data Science*. https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a Data dostępu: 2023-01-25.

[8] *The curse of dimensionality - Towards Data Science*. https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e Data dostępu: 2023-01-25.

[9] *Przestrzeń euklidesowa - Wikipedia, Wolna Encyklopedia*. https://pl.wikipedia.org/wiki/Przestrzeń_euklidesowa Data dostępu: 2023-01-27.

[10] *Principal component analysis - Towards Data Science*. https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c Data dostępu: 2023-01-27.

[11] David M. W. Powers. *Applications and Explanations of Zipf's Law*. Spraw. tech. The Flinders University of South Australia, 1998.

[12] *List of samples - Brown Corpus*. http://icame.uib.no/brown/bcm-los.html.

[13] *Introduction to Information Retrieval*. https://nlp.stanford.edu/IR-book/pdf/06vect.pdf.

[14] *TFIDF - Wikipedia, Wolna Encyklopedia*. https://pl.wikipedia.org/wiki/TFIDF.

[15] *Wyszukiwanie podobieństw semantycznych, mgr. inż. Sebastian Sitko*. https://ii.pk.edu.pl/~rkycia/master/SS.html.

[16] *Topic modeling with LSA, pLSA, LDA and word embedding - Medium*. https://medium.com/voice-tech-podcast/topic-modeling-with-lsa-plsa-lda-and-word-embedding-51bc2540b78d Data dostępu: 2023-01-27.

[17] *TruncatedSVD and its applications*. https://langvillea.people.cofc.edu/DISSECTION-LAB/Emmie'sLSI-SVDModule/p5module.html Data dostępu: 2023-01-27.

[18] *LDiA in practice - Towards Data Science*. https://towardsdatascience.com/higher-accuracy-and-less-process-time-in-text-classification-with-lda-and-tf-idf-d2d949e344c3 Data dostępu: 2023-01-16.

[19] *Dimensionality reduction for machine learning - Towards Data Science*. https://towardsdatascience.com/dimensionality-reduction-for-machine-learning-80a46c2ebb7e Data dostępu: 2023-01-16.

[20] *Word2Vec explained - Towards Data Science*. https://towardsdatascience.com/word2vec-explained-49c52b4ccb71 Data dostępu: 2023-01-17.

[21] *Backpropagation - StatQuest*. https://www.youtube.com/watch?v=IN2XmBhILt4.

[22] *A gentle introduction to Doc2Vec - Medium*. https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5el Data dostępu: 2023-01-28.

[23] *K-means clustering - scikit-learn library*. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html Data dostepu: 2023-03-20.

[24] *StatQuest: K-means clustering*. https://www.youtube.com/watch?v=4b5d3muPQmAab_channel=StatQuestwithJoshStarmer Data dostepu: 2023-03-20.

[25] *DBSCAN Parameter Estimation Using Python*. https://medium.com/@tarammullin/dbscan-parameter-estimation-ff8330e3a3bd Data dostępu: 2023-03-20.

[26] Martin Ester Hans Peter Kriegel Xiaowei Xu Erich Schubert Jorg Sander. *DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN*. Spraw. tech.

[27] *Expectation-Maximization Algorithms | Stanford CS229: Machine Learning (Autumn 2018)*. https://www.youtube.com/watch?v=rVfZHWTwXSAab_channel=StanfordOnline Data dostępu: 2023-03-20.

[28] *Gaussian Mixture - sci-kit learn library*. https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html Data dostępu: 2023-03-20.

[29] *K-means: Getting optimal number of clusters*. https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/ Data dostepu: 2023-03-20.

[30] *Gaussian Mixture Model Selection*. https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_selection.html Data dostępu: 2023-05-18.

[31] *TSNE - scikit-learn library*. https://scikitlearn.org/stable/modules/generated/sklearn.manifold.TSNE.html Data dostępu: 2023-01-30.

[32] Geoffrey Hinton Laurens van der Maaten. *Visualizing Data using t-SNE*. Spraw. tech. University of Toronto, Tilburg University, 2008.

[33] *Dimensionality reduction - TSNE*. https://apiumhub.com/tech-blog-barcelona/dimensionality-reduction-tsne/ Data dostępu: 2023-01-30.

[34] *Regular Expressions - The Python Standard Library*. https://docs.python.org/3/library/re.html.

[35] *Doc2Vec Gensim Library*. https://radimrehurek.com/gensim/models/doc2vec.html Data dostępu: 2023-01-28.

[36] *BaggingClassifier - scikit-learn library*.
`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html` Data dostępu: 2023-05-21.

[37] *StratifiedKfold - sci-kit learn library*.
`https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html` Data dostępu: 2023-05-22.

[38] *Accuracy Score - sci-kit learn library*.
`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html` Data dostępu: 2023-05-23.

[39] *Precision,recall,F1 - sci-kit learn library*.
`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html` Data dostępu: 2023-05-23.