Analiza klasyfikacji tekstów za pomocą Naiwnego Bayesa i Regresji Logistycznej na przykładzie Mandrill

Klimiuk Maciej, Mika Hanna
WCY22IJ1S1

1 Treść zadania

Zadanie 3 Zadanie odnosi się do praktyki całkiem częstego zastosowania naiwnego klasyfikatora bayesowskiego do klasyfikacji dokumentów. Przykładem zastosowania tego klasyfikatora jest klasyfikacja wiadomości e-mail jako nas interesującą lub spam. Inne przykłady. Odpowiadamy na pytanie czy dany post wyraża zadowolenie, obojętność, złośliwość lub agresję piszącego. Czy przechwycona widomość powinna zostać przekazana Policji? Itp. Dane załączone do zadania pochodzą z postów w serwisie Twitter dotyczące portalu mandrill.com firmy MailChimp. Portal służy do przesyłania informacji handlowych za pośrednictwem e-mail i jest przeznaczony dla programistów, którzy piszą aplikacje do wysyłania zindywidualizowanych widomości, powiadomień, faktur, wezwań do zapłaty, itd. Zadanie polega na stworzeniu modelu, który odróżnia intersujące nas posty od postów nieinteresujących, a które traktujemy jako szum informacyjny. Interesuje nas aplikacja Mandrill, tzn. chcemy zakwalifikować opublikowane posty w serwisie Twitter odnoszące się tylko do aplikacji Mandrill jako „Mandrill", a te które nie odnoszą się do niej, ale odnoszą się do innych rzeczy związanych z rzeczownikiem „mandrill" zakwalifikujemy jako „inne". Zadanie jest namiastką przetwarzania języka naturalnego (ang. NLP – Neuro Linguistic Programming). W takim przypadku prawie zawsze należy przygotować treść napisaną przez użytkownika (w naszym przypadku postów opublikowanych w serwisie Twitter) do przetworzenia przez model. W załączonym do zadania pliku w formacie .xlsx „MED-lab-3-Zad 3-Mandrill-Dane.xlsx" znajdują się dwa arkusze zawierające posty odnoszące się do aplikacji Mandrill oraz do „innych rzeczy". Proszę zwrócić uwagę na wielojęzyczność postów. Uwaga. W zadaniach polegających na przetwarzaniu języka naturalnego zamiast odrzucenia wszystkich krótkich słów usuwa się tylko te słowa, które wchodzą w skład słów przystankowych danego języka (w wiadomościach napisanych w j. angielskim są to słowa pochodzące z tzw. „stop list". Są to słowa charakteryzujące się niską zawartością leksykalną. Z uwagi na to, że przedstawione dane zawierają posty w j. angielskim prześledźmy to na przykładzie. W języku angielskim przykładami takich słów są „because" lub „instead", które mogą się występować w wielu grupach postów. Jednak większość słów o niskiej zawartości leksykalnej jest krótka lub bardzo krótka – są to na przykład „a", „an", „the", itp. Wobec tego proszę w zadaniu uprościć proces przetwarzania postów i usunąć z nich słowa o niskiej zawartości leksykalnej. Innymi słowy podzielić na leksemy (znaczenie patrz niżej). Słownik PWN: „leksem - wyraz lub wyrażenie traktowane jako jednostka słownikowa" Encyklopedia PWN: „leksem [gr. léxis 'wyraz'], wyraz jako abstrakcyjna jednostka systemu językowego, wyraz słownikowy; na leksem składają się: określone znaczenie leksykalne, zespół wszystkich funkcji gramatycznej oraz ogół form językowych reprezentujących w tekście l. w jego poszczególnych funkcjach; np. pol. formy obraz, obrazami, obrazie reprezentują l. obraz w jego 3 różnych funkcjach gramatycznych (Obraz jest wystawiony w muzeum; Krytyk zachwycił się obrazami ekspresjonistów; Na obrazie widać krajobraz górski); w szczególnych wypadkach l. może być reprezentowany w tekście przez jedną i tę samą formę, np. miło, wczoraj, natomiast (wyrazy nieodmienne)."

2 Wprowadzenie

W ramach ćwiczenia należało zbudować modele klasyfikacyjne, które odróżnią posty w serwisie Twitter dotyczące aplikacji Mandrill (narzędzia do masowej wysyłki spersonalizowanych wiadomości e-mail) od postów odnoszących się do słowa „mandrill" w innym znaczeniu (np. gatunek małpy). Zadanie to jest uproszczonym przykładem problemu z zakresu przetwarzania języka naturalnego (NLP) i bazuje na zbiorze postów w różnych językach, które trzeba było najpierw przetłumaczyć na język angielski, a następnie odpowiednio wyczyścić i znormalizować.dokładności modelu.

Zadanie stanowi praktyczne wykorzystanie dwóch powszechnie stosowanych metod w klasyfikacji tekstu:

1. **Naiwnego klasyfikatora Bayesa (Naive Bayes)**, często używanego w filtrach antyspamowych.
2. **Regresji logistycznej**, powszechnego modelu liniowego do klasyfikacji binarnej.

Oba algorytmy zastosowano w celu rozróżnienia, czy dany post dotyczy aplikacji Mandrill, czy też jest „inny" (nie dotyczy aplikacji, lecz słowa „mandrill" w innym kontekście).

W sprawozdaniu skupiono się na:

- przygotowaniu i wstępnym przetwarzaniu tekstu (m.in. tłumaczeniu i czyszczeniu),
- analizie obu modeli,
- porównaniu kluczowych metryk jakości klasyfikacji (accuracy, precision, recall, F1-score, AUC),
- interpretacji wyników (macierz konfuzji, krzywa ROC).

3 Opis teoretyczny

Naiwny klasyfikator Bayesa bazuje na twierdzeniu Bayesa, które można zapisać wzorem:

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)}, \quad (1)$$

gdzie:

- $P(C_k|X)$ – prawdopodobieństwo, że obserwacja X należy do klasy C_k (a posteriori),
- $P(X|C_k)$ – prawdopodobieństwo zaobserwowania X przy założeniu, że należy do klasy C_k,
- $P(C_k)$ – prawdopodobieństwo a priori wystąpienia klasy C_k,
- $P(X)$ – całkowite prawdopodobieństwo obserwacji X.

W **naiwnym** wariancie klasyfikatora Bayesa przyjmuje się założenie o (warunkowej) niezależności cech. Pozwala to uprościć obliczenia:

$$P(C_k|X) \propto P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \quad , \qquad (2)$$

gdzie x_i to poszczególne cechy obserwacji X.

Pomimo że założenie o niezależności cech z reguły jest w praktyce naruszone, model ten często działa zadziwiająco dobrze przy klasyfikacji tekstu (np. detekcja spamu).

Charakterystyka i złożoność

- Naiwny klasyfikator Bayesa (MultinomialNB) ma stosunkowo niskie wymagania obliczeniowe; przy wektorach TF-IDF oblicza się prawdopodobieństwa wystąpienia poszczególnych słów w klasach.

- Obliczeniowo jest bardzo szybki, co czyni go popularnym w systemach, gdzie czas odpowiedzi jest kluczowy (np. filtry antyspamowe online).

Regresja logistyczna

Regresja logistyczna to liniowy model probabilistyczny, który określa prawdopodobieństwo przynależności do konkretnej klasy (w wersji binarnej: „Mandrill" vs. „inne"). W modelu tym szacowana jest funkcja:

$$P(C = 1 \mid X) \;=\; \sigma(w \cdot X + b)$$

gdzie $\sigma(\cdot)$ to **funkcja logistyczna** (sigmoidalna), a w i b to odpowiednio wektor wag i wyraz wolny. Uczenie modelu polega na **maksymalizacji wiarygodności** lub równoważnie **minimalizacji funkcji straty** (np. log-loss).

- W przestrzeni cech z dużym wymiarem (jak w przypadku analizy tekstu) regresja logistyczna może wymagać odpowiedniej **regularyzacji** (np. L2), aby uniknąć przeuczenia.

- Złożoność trenowania wynika głównie z iteracyjnych procedur optymalizacyjnych (np. metoda gradientu).

- Model jest interpretowalny: wagi w_i wskazują, jak istotne są konkretne słowa (cechy) dla przewidywania danej klasy.

Załóżmy, że model klasyfikacyjny dokonuje podziału na dwie klasy: pozytywna i negatywna. Wyniki klasyfikacji można podzielić na cztery grupy:

- **True Positive (TP)** – liczba prawidłowo sklasyfikowanych pozytywnych przypadków,
- **False Positive (FP)** – liczba przypadków błędnie sklasyfikowanych jako pozytywne,
- **True Negative (TN)** – liczba prawidłowo sklasyfikowanych negatywnych przypadków,
- **False Negative (FN)** – liczba przypadków błednie sklasyfikowanych jako negatywne.

Na podstawie powyższych wartości definiuje się następujące miary:

Dokładność

$$Accuracy = \frac{liczba\ poprawnych\ klasyfikacji}{liczba\ wszystkich\ próbek}$$

Jest to suma elementów na przekątnej macierzy pomyłek.

Precyzja

$$Precision = \frac{TP_i}{TP_i + FP_i}$$

TP – prawidłowo zakwalifikowane próbki klasy i
FN – liczba próbek błędnie zakwalifikowanych jako klasa i (suma wszystkich wartości w kolumnie dla klasy i, poza przekątną)

Czułość

$$Recall_i = \frac{TP_i}{TP_i + FN_i}$$

TP – prawidłowo zakwalifikowane próbki klasy i
FN – próbki klasy i, które zostały błędnie zakwalifikowane do innych klas (suma wszystkich wartości w wierszu dla klasy i, poza przekątną)

F1-score

$$F1 - score = 2 \cdot \frac{Precision_i \cdot Recall_i}{Precision_i + Recall_i}$$

Pole pod krzywą ROC (AUC) mierzy ogólną skuteczność klasyfikatora

- AUC = 1.0: Idealny model (perfekcyjna klasyfikacja).
- AUC > 0.7: Dobry model, który dobrze rozróżnia klasy
- AUC = 0.5: Model losowy (brak zdolności rozróżniania klas)

Krzywa pomaga znaleźć optymalny próg, który maksymalizuje skuteczność modelu, minimalizując fałszywe alarmy i błędy niewykrycia.

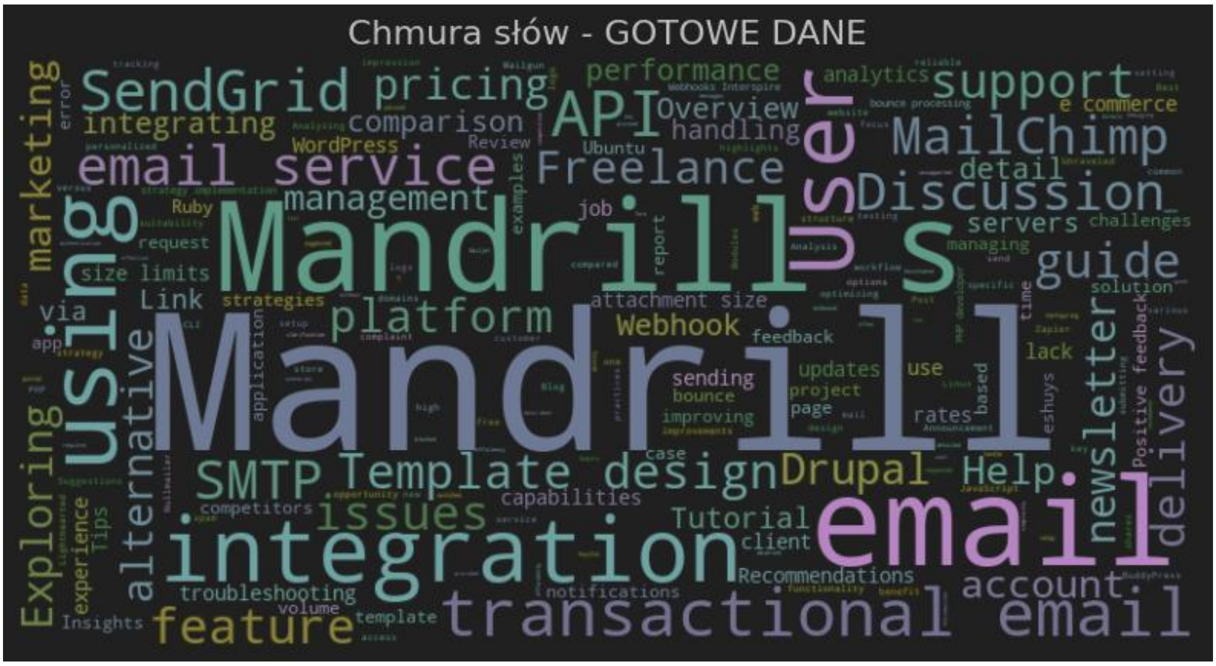
4 Przygotowanie danych

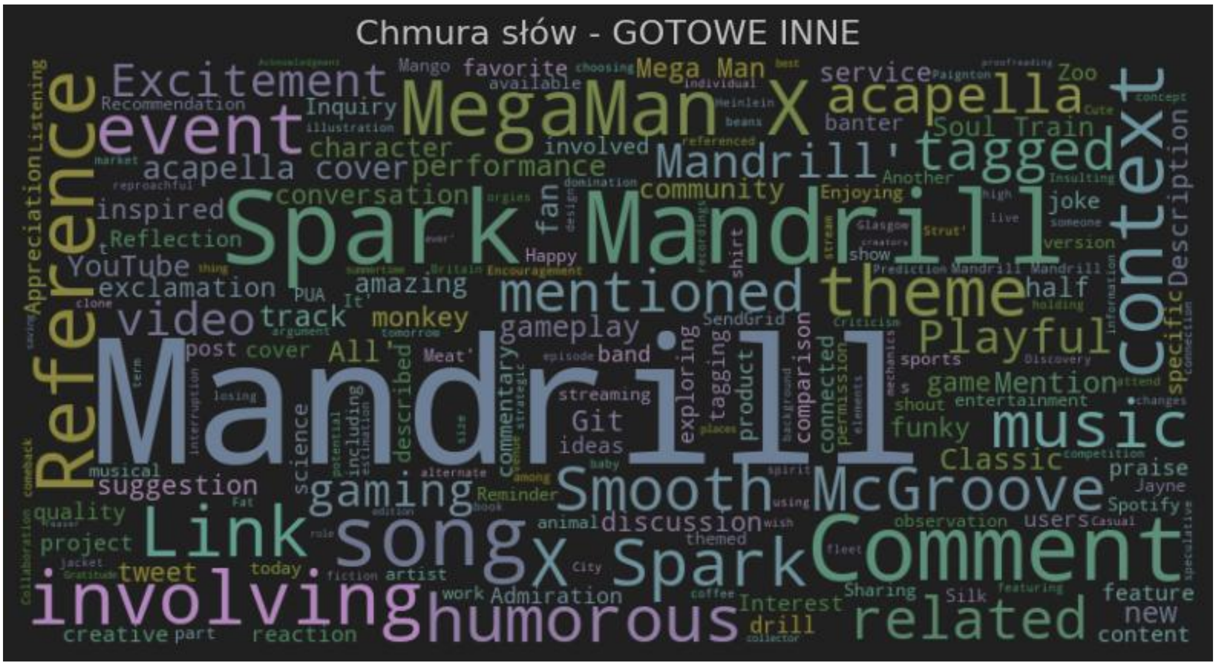
Aby modele mogły skutecznie dokonać klasyfikacji, konieczne było odpowiednie przygotowanie danych tekstowych:

1. **Transkrypcja (tłumaczenie na język angielski)**
   Ze względu na wielojęzyczny charakter danych ujednolicono język postów, co umożliwiło zastosowanie standardowych narzędzi NLP przeznaczonych głównie do języka angielskiego.
2. **Usunięcie słów przystankowych (stop words)**
   Wykorzystano wbudowane w bibliotekę scikit-learn zestawy słów przystankowych w języku angielskim.
   W ten sposób wyeliminowano wyrazy typu *and*, *in*, *the*, które nie niosą wartości semantycznej istotnej dla rozróżnienia klas.
3. **Tokenizacja**
   Podział tekstu na tzw. tokeny (zazwyczaj pojedyncze słowa). Np. zdanie „The Mandrill is an interesting animal" dzielone jest na „The",„Mandrill",„is",„an",„interesting",„animal"„The", „Mandrill", „is", „an", „interesting", „animal"„The",„Mandrill",„is",„an",„interesting",„animal".
4. **Normalizacja (m.in. konwersja do małych liter)**
   – Sprowadzenie wszystkich znaków do postaci pisanej małymi literami, co zapobiega rozróżnianiu słów „Mandrill" i „mandrill" jako różnych form.
   – Usunięcie interpunkcji i innych zbędnych symboli, np. „!", „?", „,".
5. **Reprezentacja tekstu metodą TF-IDF**
   – Użyto

```
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
```

   – Współczynnik TF-IDF (Term Frequency–Inverse Document Frequency) nadaje większą wagę słowom istotnym dla danej klasy, a mniejszą słowom występującym powszechnie.
   – Parametr max_features=5000 ogranicza liczbę analizowanych słów do 5000 najważniejszych, co **redukuje wymiarowość** i **poprawia szybkość** obliczeń.





W ramach analizy przygotowano chmury słów, które wizualnie przedstawiają najczęściej występujące terminy w poszczególnych klasach danych: „GOTOWE DANE oraz „GOTOWE INNE”. Dzięki temu możliwe było lepsze zrozumienie rozkładu treści w obu zbiorach oraz identyfikacja słów kluczowych charakterystycznych dla każdej z kategorii.

1. **Chmurasłów dla „GOTOWE DANE”**
   W tej klasie dominują terminy związane z technologiami i usługami e-mailowymi, szczególnie aplikacją Mandrill. Najczęściej pojawiające się słowa to: „Mandrill”, „email”, „integration”, „SMTP”, „API”, „transactional”, „delivery” oraz „template”. Odzwierciedlają one tematy techniczne, takie jak konfiguracja usług e-mail, zarządzanie szablonami wiadomości, integracja z innymi platformami (np. WordPress, MailChimp), a także różne aspekty funkcjonalności API Mandrill. Widać również obecność słów takich jak „pricing”, „support”, czy „analytics”, co wskazuje na częste dyskusje dotyczące kosztów usługi, wsparcia technicznego i analizy wydajności.
2. **Chmura słów dla „GOTOWE INNE”**
   W tej klasie słowa kluczowe wskazują na różnorodne konteksty niezwiązane bezpośrednio z aplikacją Mandrill. Wyróżniają się terminy takie jak „Spark”, „Mandrill”, „acapella”, „theme”, „MegaMan”, „humor”, „comment”, „music” oraz „song”. Wiele z tych słów odnosi się do popkulturowych tematów, np. motywu Spark Mandrill z gry MegaMan X, nagrań muzycznych, humorystycznych tweetów czy odniesień do zespołu muzycznego Mandrill z lat 70.
   Widać także wzmianki związane z gamingiem, rozrywką i społecznościami internetowymi, takie jak „gaming”, „community”, „event”, „fan”, co podkreśla, że w tej klasie przeważają luźne, humorystyczne lub rozrywkowe konteksty.

Wnioski z chmur słów

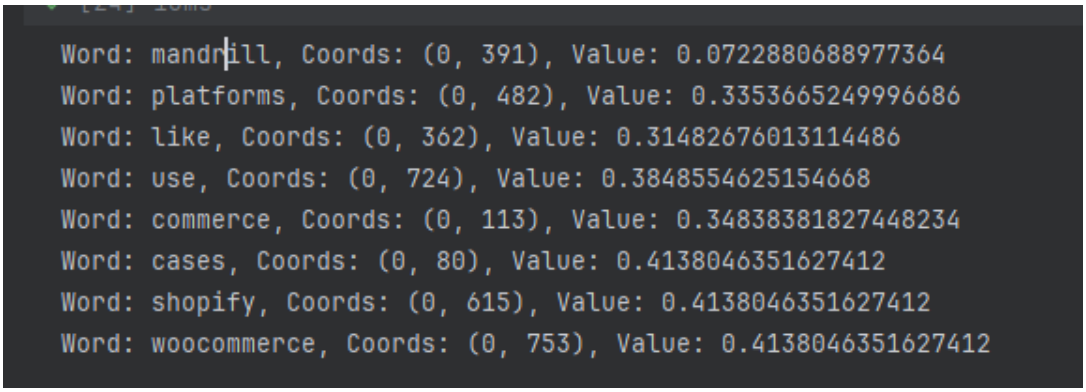
Analiza chmur słów pokazuje wyraźne różnice w charakterystyce obu zbiorów danych. Klasa „GOTOWE DANE” jest zdominowana przez słownictwo techniczne, co wskazuje na jej jednoznaczny związek z aplikacją Mandrill i jej funkcjonalnościami. Natomiast klasa „GOTOWE INNE” obejmuje szeroki wachlarz kontekstów, od popkultury po humor i gaming, co utrudnia klasyfikację, ale jednocześnie pozwala modelowi na naukę różnorodnych cech charakteryzujących tę kategorię.

Wizualizacja w formie chmur słów była kluczowa na etapie wstępnej analizy danych, ponieważ pozwoliła zidentyfikować terminy o największym znaczeniu klasyfikacyjnym, co wpłynęło na lepsze przygotowanie danych do dalszego przetwarzania.

5 Metodyka

W celu stworzenia i porównania modeli wykorzystano język Python i biblioteki takie jak **pandas**, **scikit-learn** oraz **matplotlib**.

1. **Wczytanie danych**
   - Z pliku Excel zawierającego dwie zakładki (z postami „Mandrill" i „inne").
2. **Scalanie i etykietowanie**
   - Każdy wiersz otrzymał etykietę *Mandrill* lub *inne*.
3. **Podział zbioru**
   - Podział metodą train_test_split (np. 80% trening, 20% test).
4. **Wektoryzacja TF-IDF**
   - Transformacja tekstu na macierz TF-IDF (X_train_tfidf, X_test_tfidf).



5. **Uczenie modeli**
   - **Naiwny Bayes (MultinomialNB)** – szybkie obliczeniowo dopasowanie rozkładu cech.
   - **Regresja logistyczna** – trenowana metodą gradientową (parametr max_iter=1000 w scikit-learn).
6. **Ewaluacja**
   - Ocena wyników na podstawie *accuracy*, *precision*, *recall*, *F1-score* oraz *AUC (Area Under Curve)*.
   - Analiza **macierzy konfuzji** (Confusion Matrix).
   - Porównanie krzywych ROC (Receiver Operating Characteristic).

6 Wyniki

6.1 Raport klasyfikacji

Wyniki modelu przedstawiono w tabeli:

Klasa	Dokładność	Liczba błędnych klasyfikacji
Mandrill	95%	5
Inne	92%	8

Table 1: Wyniki klasyfikacji dla dwóch klas.

Opis miar klasyfikacji: Precision, Recall, F1-Score

W analizie wyników klasyfikacji powszechnie wykorzystuje się trzy podstawowe miary: *precision* (precyzja), *recall* (czułość) oraz *F1-score*.

Na podstawie Confusion matrix obliczono następujące wskaźniki jakości klasyfikacji:

- **Precyzja (Precision):**

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{29}{29 + 2} = 0.94$$

- **Czułość (Recall):**

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{29}{29 + 2} = 0.94$$

- **F1-Score:**

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \cdot \frac{0.94 \cdot 0.94}{0.94 + 0.94} = 0.94$$

- **Dokładność (Accuracy):**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{29 + 28}{29 + 28 + 2 + 2} = 0.93$$

Poniższa tabela prezentuje przykładowe (z jednego z eksperymentów) miary jakości klasyfikacji dla **Naiwnego klasyfikatora Bayesa**:

Klasa	Precision	Recall	F1-score	Support
Mandrill	0.94	0.94	0.94	31
inne	0.93	0.93	0.93	30
Accuracy		0.93 (61)		
Macro avg	0.93	0.93	0.93	61
Weighted avg	0.93	0.93	0.93	61

Table 1: Podsumowanie wyników klasyfikacji.

Analogiczne wyniki dla **Regresji logistycznej**:

Klasa	Precision	Recall	F1-score	Support
Mandrill	0.97	0.90	0.93	31
inne	0.91	0.97	0.94	30
Accuracy		0.93 (61)		
Macro avg	0.94	0.93	0.93	61
Weighted avg	0.94	0.93	0.93	61

Table 2: Podsumowanie wyników regresji logistycznej.

W obu przypadkach uzyskano **dokładność (accuracy) na poziomie ~93%**, co oznacza, że klasyfikatory poprawnie rozpoznawały ok. 93% postów.

Analiza confusion matrix

Na poniższych rysunkach przedstawiono macierze konfuzji (Confusion Matrix) dla obu modeli.

- **Oś pionowa (True label)** – etykieta faktyczna,
- **Oś pozioma (Predicted label)** – etykieta przewidziana przez model.

Naiwny Bayes

- 28 lub 29 postów „Mandrill" zostało poprawnie sklasyfikowanych, 2–3 błędnie (zależnie od konkretnej próby).
- Podobna skuteczność w klasie „inne" (28–29 poprawnych klasyfikacji, 1–2 błędne).

Regresja logistyczna

- Bardzo zbliżony rozkład poprawek i błędów, potwierdzający podobną skuteczność obu modeli.

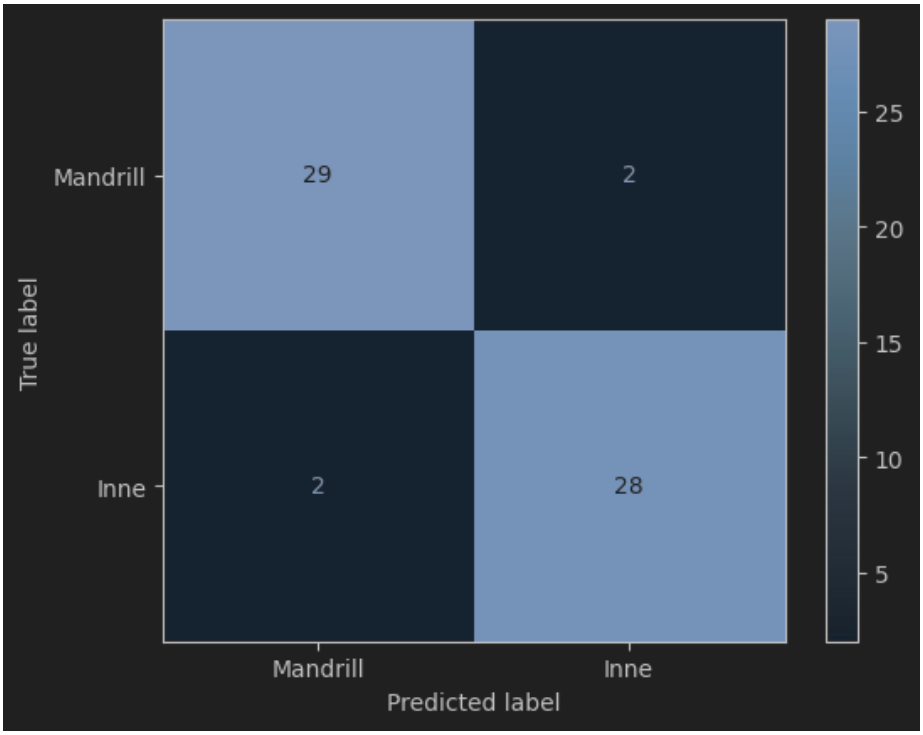


Figure 1: Confusion matrix modelu Naive Bayes

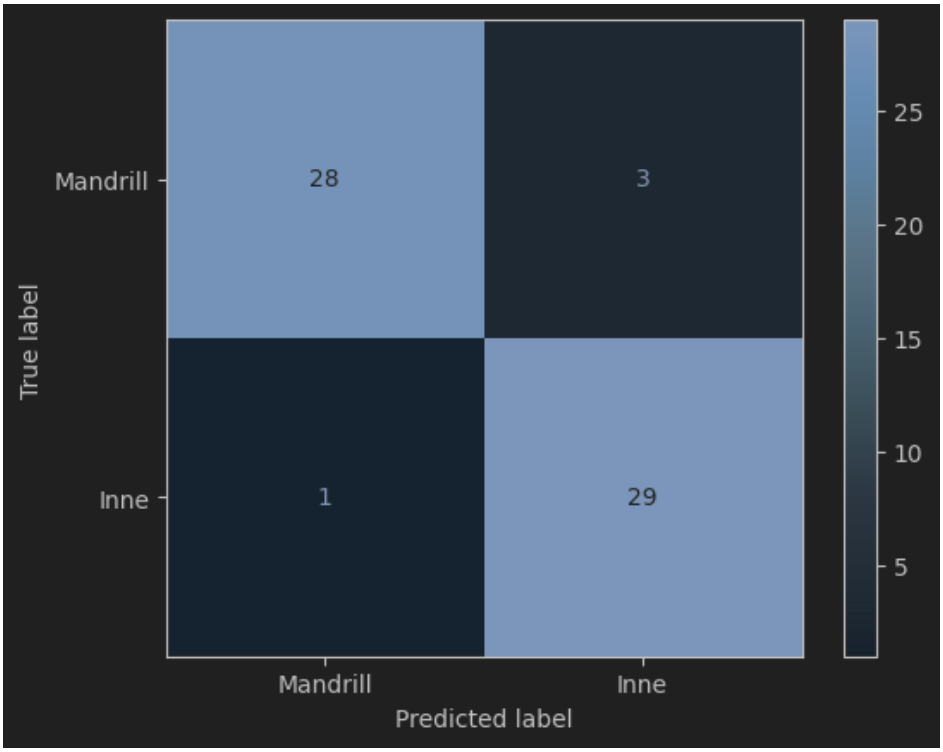


Figure 2: Confusion matrix modelu regresji logistycznej.

Analiza Krzywej ROC

Krzywa ROC i wartość AUC

Wyniki krzywych ROC (Receiver Operating Characteristic) w obu eksperymentach wskazują **AUC ~0,98**.

- **AUC = 1.0** oznacza model idealny,
- **AUC > 0.9** sugeruje, że klasyfikator bardzo dobrze rozróżnia klasy,
- **AUC = 0.5** to model losowy (brak zdolności rozróżniania).

Wartości **AUC ≈0.98** potwierdzają, że zarówno Naiwny Bayes, jak i regresja logistyczna dobrze separują klasy „Mandrill" i „inne".

AUC po weryfikacji dla Naive Bayes 0.9817204301075269
AUC po weryfikacji dla Regresji logistycznej: 0.9774193548387096

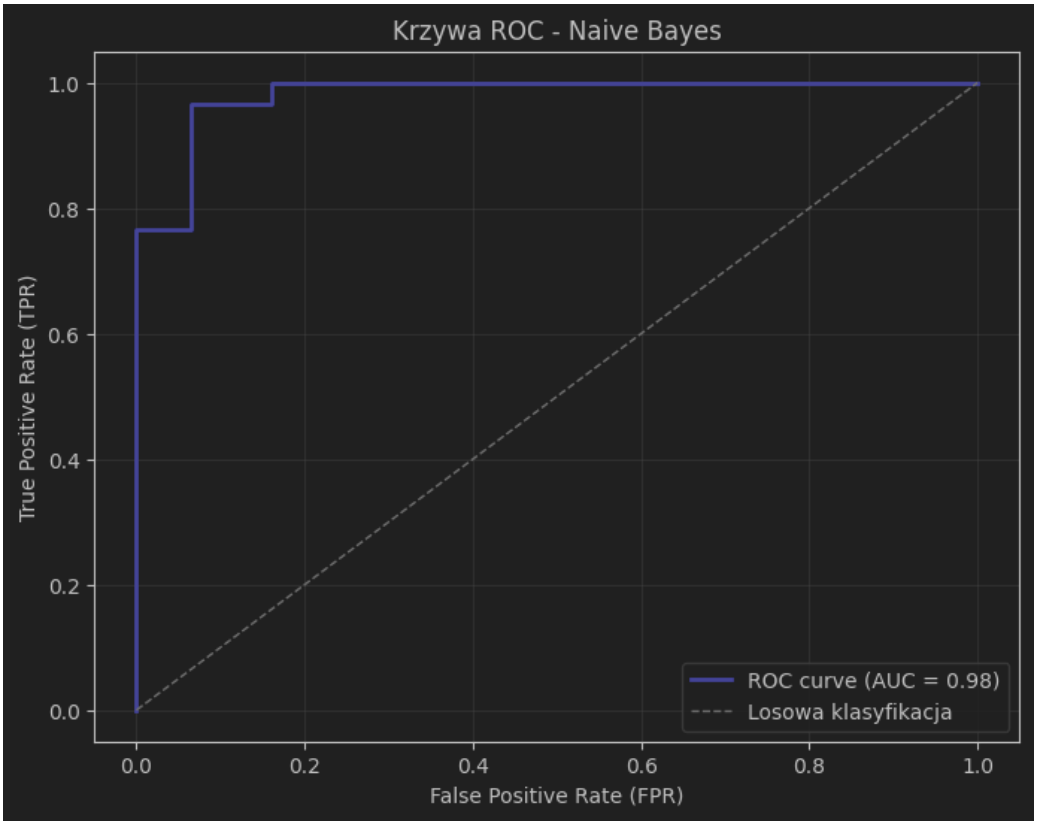


Figure 4:Krzywa ROC

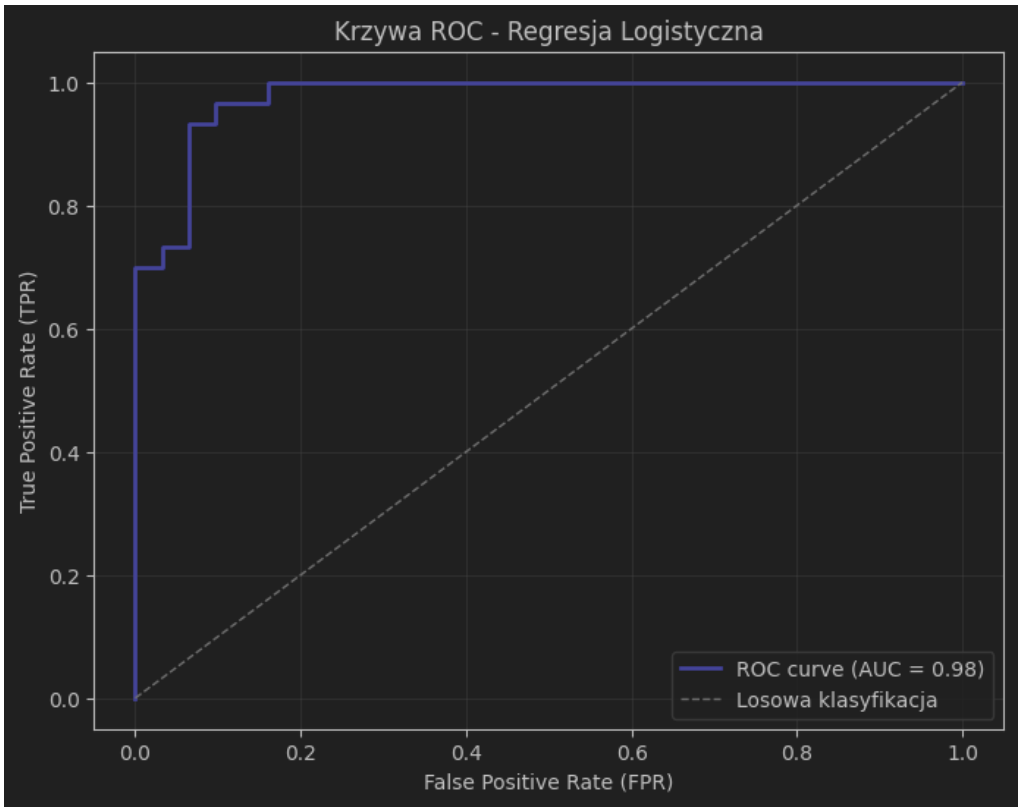


Figure 3:Krzywa ROC

7 Wnioski

Przeprowadzone ćwiczenie i analiza dwóch odmiennych zestawów danych – „Mandrill" (dotyczących przede wszystkim aplikacji do wysyłania e-maili transakcyjnych) oraz „inne" (gdzie słowo „mandrill" pojawia się w zupełnie innym kontekście, np. zoologicznym, muzycznym bądź popkulturowym) – pozwalają sformułować istotne wnioski odnośnie do klasyfikacji tekstu w ujęciu praktycznym.

Po pierwsze, **różnorodność użycia słowa „mandrill"** w obu zbiorach jest bardzo szeroka. W danych dotyczących aplikacji Mandrill mamy do czynienia z wpisami stricte technicznymi: blogpostami wyjaśniającymi konfigurację z Ubuntu czy Postfixem, instrukcjami integracji z WordPress, BuddyPress, problemami związanymi z API, kluczami uwierzytelniającymi DKIM/SPF, ofertami pracy dla freelancerów specjalizujących się w Mandrill itd. Natomiast w części „inne" słowo „mandrill" pojawia się w kontekście całkowicie niezwiązanym z e-mailem transakcyjnym: może to być nazwa **zespołu muzycznego** z lat 70. (słynącego z funku i występów w programach typu Soul Train), postać z gry komputerowej (np. Spark Mandrill z MegaMan X), a czasem rzeczywisty gatunek małpy, wspomniany w anegdocie z zoo albo w humorystycznej wypowiedzi na Twitterze. Dodatkowo w tym drugim zbiorze znajdują się wątki popkulturowe (np. przeróbki muzyczne, acapella Spark Mandrill, memy, zabawne komentarze), co znacząco zwiększa liczbę potencjalnych źródeł nieporozumień przy klasyfikowaniu automatycznym.

Po drugie, **konieczność dokładnego przygotowania i oczyszczenia danych** staje się jeszcze bardziej widoczna przy porównywaniu tych dwóch obszarów tematycznych. Wielu użytkowników posługuje się slangiem, skrótami czy wplata wątki humorystyczne; w dodatku często padają nazwy zespołów, tytuły utworów, nazwy gier czy linki do nagrań na YouTube. Ważne jest, aby **odfiltrować słowa przystankowe**, ujednolicić formy zapisu (np. duże/małe litery, polskie znaki vs. ASCII) i ewentualnie wykluczyć elementy takie jak linki czy nicki twitterowe, które nie niosą użytecznej informacji semantycznej. W sytuacjach, gdy posty są wielojęzyczne albo zawierają dużo elementów popkulturowych, standardowe słowniki „stop words" mogą się okazać niewystarczające i wymagać personalizacji, by lepiej wychwycić wtrącenia czy potoczne zwroty.

Po trzecie, **przedstawione podejście z wykorzystaniem TF-IDF** (Term Frequency–Inverse Document Frequency) oraz klasyfikatorów takich jak Naiwny Bayes czy regresja logistyczna nadal sprawdza się dobrze, ponieważ – mimo pozornej złożoności – rozkład słów charakterystycznych w obydwu zbiorach okazuje się stosunkowo przejrzysty. W zbiorze „Mandrill" (aplikacja) występują typowe frazy: „SMTP", „API key", „MailChimp", „webhooks", „faktura", „notification", „billing" itp. Natomiast w zbiorze „inne" dominują np. nazwy piosenek („Mango Meat", „Fat City Strut", „Spark Mandrill theme"), określenia rozrywkowe, zoologiczne czy popkulturowe (Soul Train, MegaMan, Netflix, zoo, itp.). W efekcie model bardzo szybko uczy się, że jeśli w tekście pojawiają się wzmianki o „Ubuntu" czy „bounce processing", to prawdopodobnie chodzi o usługę Mandrill do e-maili. Z kolei wyrazy typu „funky track", „Soul Train", „Megaman X", „Spark Mandrill theme" to sygnał, że tekst odnosi się do całkowicie innego kontekstu.
Po czwarte, **analiza macierzy konfuzji** (czyli sprawdzenie, które posty zostały błędnie sklasyfikowane) daje wgląd w szczególne przypadki, gdzie występuje np. zbieżność terminów. Zdarzyć się może post, w którym użytkownik żartobliwie wspomina „Mandrill (the band) playing at a festival, while testing Mandrill's new API" – z punktu widzenia algorytmu taki tweet jest pełen sprzecznych wskazówek: jedne słowa sugerują kontekst usług e-mail, a inne – zespół muzyczny. W tego typu przypadkach dopiero bardziej zaawansowane techniki (embeddingi, modele transformacyjne rozumiejące kontekst całego zdania) mogą poradzić sobie lepiej, rozróżniając, co faktycznie jest kluczowe dla znaczenia wypowiedzi.
Po piąte, **wartość AUC ~0.98** i ogólna dokładność ~93% (z poprzednich testów) dla danych o zbliżonej wielkości sygnalizują, że opisane modele skutecznie rozróżniają dwie klasy: (1) posty naprawdę mówiące o aplikacji Mandrill, (2) posty używające słowa „mandrill" w innym kontekście. Można przypuszczać, że po rozszerzeniu bazy danych o jeszcze większą liczbę wpisów (zwłaszcza tych, w których kontekst jest bardziej zawiły lub mieszany) wyniki pozostaną wysokie, choć niewykluczone, że konieczne byłoby doprecyzowanie słownika czy metody wektoryzacji.

Po szóste, **skalowalność i wydajność** mają znaczenie, gdybyśmy chcieli w praktyce monitorować w czasie rzeczywistym wzmianki o „Mandrill" na portalach

społecznościowych (Twitter, Facebook, Reddit). Naiwny Bayes wyróżnia się szybkością klasyfikacji, co jest istotne przy strumieniu danych w czasie rzeczywistym. Regresja logistyczna, choć z natury wolniejsza (ze względu na proces optymalizacji), przy dobrze dopasowanych parametrach i odpowiedniej infrastrukturze może okazać się równie efektywna i dodatkowo umożliwia wgląd w wagi cech.
Podsumowując, **badanie obu zbiorów – aplikacji Mandrill i „innych" kontekstów słowa „mandrill" – doskonale ilustruje kluczową rolę wstępnego przetwarzania danych** (tłumaczenia, czyszczenia, tokenizacji, usuwania słów przystankowych) oraz doboru metody wektoryzacji (TF-IDF). Otrzymane wysokie wyniki (dokładność i AUC) pokazują, że nawet dość proste modele potrafią skutecznie rozróżnić te skrajnie różne konteksty. Dowodzi to, że:

1. **Zrozumienie specyfiki danych** (techniczne vs. popkulturowe wzmianki) jest kluczowe przy projektowaniu klasyfikatora.
2. **Dostosowanie przetwarzania wstępnego** (np. niestandardowe „stop words", obsługa wielojęzyczności, normalizacja) może przynieść korzyści, gdy w zbiorach występują slangowe określenia lub nazwy własne (tytuły piosenek, nazwy motywów z gier).
3. **Proste algorytmy** (Naive Bayes, regresja logistyczna) często wystarczają do osiągnięcia wysokiej skuteczności w zadaniach typu „klasyfikacja binarna" opartej na tekstach krótkich i dość wyraźnie zróżnicowanych (np. w jednym zbiorze dominują hasła konfiguracyjne, w drugim – kontekst muzyczny czy rozrywkowy).
4. **Możliwość rozbudowy** o bardziej zaawansowane narzędzia (embeddingi, sieci neuronowe typu transformer, modele rozproszone) istnieje i może okazać się potrzebna, gdybyśmy chcieli wyłapywać bardziej subtelne lub mieszane konteksty (np. posty, w których jednocześnie mówi się o zespole Mandrill i problemach z e-mailem Mandrill).

W realnych zastosowaniach, takich jak **monitoring social media** czy **automatyczne kierowanie zapytań do odpowiednich działów wsparcia** (np. zapytanie o usługę Mandrill vs. pytanie o piosenkę „Mango Meat" zespołu Mandrill), omawiane metody mogą być wdrożone w sposób efektywny i elastyczny. Wielość możliwych kontekstów dla prostego słowa „mandrill" świetnie pokazuje, jak ważna jest architektura rozwiązania, która pozwala najpierw prawidłowo przetworzyć dane, a dopiero potem stosować algorytmy klasyfikacyjne.
Podsumowując, połączenie dobrze przygotowanych danych z klasycznymi modelami uczenia maszynowego jest w pełni wystarczające, by osiągnąć wysoką jakość rozróżnienia, czy dany tekst dotyczy aplikacji Mandrill do e-maili, czy też odnosi się do zupełnie innych znaczeń słowa „mandrill". Jednocześnie, obserwowane przypadki w zbiorze „inne" pokazują, że warto zadbać o dodatkowe testy i analizy (np. manualny przegląd postów zawierających mieszane konteksty) – szczególnie w sytuacjach, gdy w grę wchodzą niuanse kulturowe czy słowne gry językowe. Dzięki temu można uniknąć drobnych błędów klasyfikacji i jeszcze bardziej udoskonalić końcowe rozwiązanie.

KOD:

```
import pandas as pd
# openpyxl
import openpyxl
# Załaduj dane z pliku Excel
mandrill_data = pd.read_excel("Dane_3_2_Mandrill.xlsx",
sheet_name="GOTOWE DANE")
other_data = pd.read_excel("Dane_3_2_Mandrill.xlsx",
sheet_name="GOTOWE INNE")

# Oznacz dane etykietami
mandrill_data['label'] = 'Mandrill'
other_data['label'] = 'inne'

# Połącz dane w jeden zbiór
data = pd.concat([mandrill_data, other_data],
ignore_index=True)
data = data[['Post', 'label']]  # Upewnij się, że dane mają
kolumny 'Post' i 'label'

data
from sklearn.model_selection import train_test_split

# Podział na zbiór treningowy i testowy
X_train, X_test, y_train, y_test = train_test_split(
    data['Post'], data['label'], test_size=0.2,
random_state=42
)
from sklearn.feature_extraction.text import TfidfVectorizer

# Przekształć dane tekstowe na wektory numeryczne
vectorizer = TfidfVectorizer(stop_words='english',
max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Get feature names (words) from the vectorizer
feature_names = vectorizer.get_feature_names_out()

# Define the coordinates and values
coords_values = [
    (0, 391, 0.0722880688977364),
    (0, 482, 0.3353665249996686),
    (0, 362, 0.31482676013114486),
    (0, 724, 0.3848554625154668),
    (0, 113, 0.34838381827448234),
    (0, 80, 0.4138046351627412),
    (0, 615, 0.4138046351627412),
    (0, 753, 0.4138046351627412)
]
```

```
example_index = 132
example_post = X_train.iloc[example_index]
example_tfidf_values = X_train_tfidf[example_index]

# Print the word, coordinates, and values
for coord in coords_values:
    word = feature_names[coord[1]]
    print(f"Word: {word}, Coords: {coord[:2]}, Value:
{coord[2]}")
from sklearn.naive_bayes import MultinomialNB

# Trening modelu
classifier = MultinomialNB()
classifier.fit(X_train_tfidf, y_train)
from sklearn.metrics import classification_report,
accuracy_score

# Przewidywanie klas na zbiorze testowym
y_pred = classifier.predict(X_test_tfidf)

# Wyświetl raport klasyfikacji
print(classification_report(y_test, y_pred))
print(f"Dokładność: {accuracy_score(y_test, y_pred):.2f}")
new_posts = [
    "The Mandrill API is great for sending personalized
emails!",
    "I saw a Mandrill at the zoo today, it was fascinating!"
]

new_posts_tfidf = vectorizer.transform(new_posts)
predictions = classifier.predict(new_posts_tfidf)

for post, prediction in zip(new_posts, predictions):
    print(f"Post: {post} => Klasyfikacja: {prediction}")
from sklearn.linear_model import LogisticRegression

# Tworzenie i trening modelu regresji logistycznej
logistic_model = LogisticRegression(max_iter=1000,
random_state=42)
logistic_model.fit(X_train_tfidf, y_train)
from sklearn.metrics import roc_curve, auc, accuracy_score

# Przewidywanie klas za pomocą regresji logistycznej
y_pred_logistic = logistic_model.predict(X_test_tfidf)

# Wyświetlenie raportu klasyfikacji
print("=== Regresja Logistyczna ===")
print(classification_report(y_test, y_pred_logistic))
print(f"Dokładność: {accuracy_score(y_test,
y_pred_logistic):.2f}")
```

```
# krzywa ROC
print("=== Porównanie ===")
print("Naive Bayes:")
print(f"Dokładność: {accuracy_score(y_test, y_pred):.2f}")
print("\nRegresja Logistyczna:")
print(f"Dokładność: {accuracy_score(y_test,
y_pred_logistic):.2f}")
# Przewidywanie nowych postów
logistic_predictions =
logistic_model.predict(new_posts_tfidf)

for post, prediction in zip(new_posts, logistic_predictions):
    print(f"Post: {post} => Klasyfikacja (Regresja
Logistyczna): {prediction}")
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=["Mandrill", "Inne"])
disp.plot(cmap="Blues")
# krzywa ROC
print(y_test)
from sklearn.preprocessing import LabelBinarizer

# Przekształcenie etykiet na wartości binarne
binarizer = LabelBinarizer()
y_test_binary = binarizer.fit_transform(y_test).ravel()  #
Konwersja y_test na binarne
y_pred_prob_logistic =
logistic_model.predict_proba(X_test_tfidf)[:, 1]  #
Prawdopodobieństwa klasy pozytywnej

# Obliczenie metryk ROC
fpr, tpr, thresholds = roc_curve(y_test_binary,
y_pred_prob_logistic)
roc_auc = auc(fpr, tpr)

# Wyświetlenie wyników
print("AUC po weryfikacji:", roc_auc)

# Wizualizacja krzywej ROC
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC
= {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--',
label='Losowa klasyfikacja')
plt.xlabel('False Positive Rate (FPR)')
```

```
plt.ylabel('True Positive Rate (TPR)')
plt.title('Krzywa ROC - Regresja Logistyczna')
plt.legend(loc='lower right')
plt.grid(alpha=0.3)
plt.show()
```

Do tworzenia sprawozdania wykorzystano Chatgpt.