WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ CYBERNETYKI

SPRAWOZDANIE
Metody Eksploracji Danych

Temat laboratorium: **KLASYFIKACJA POSTÓW NA PODSTAWIE NAIWNEGO KLASYFIKATORA BAYESOWSKIEGO**

INFORMATYKA

(kierunek studiów)

INŻYNIERIA SYSTEMÓW – ANALIZA DANYCH

(specjalność)

Zespół:

Michał ŚLĘZAK
Szymon OLEŚKIEWICZ

Prowadzący laboratorium:

Dr inż. Romuald Hoffmann, prof. WAT

Warszawa 2025

Spis treści

Wstęp

Celem niniejszego sprawozdania jest prezentacja praktycznego zastosowania naiwnego klasyfikatora Bayesa do problemu klasyfikacji postów pochodzących z serwisu Twitter. Zadanie polega na automatycznym rozróżnianiu postów odnoszących się do aplikacji Mandrill (narzędzia do wysyłania wiadomości e-mail firmy MailChimp) od postów dotyczących innych znaczeń słowa "mandrill" (np. gatunku małpy).

Problem stanowi przykład zastosowania metod przetwarzania języka naturalnego (NLP) w eksploracji danych. Klasyfikacja dokumentów tekstowych jest jednym z najczęstszych zastosowań naiwnego klasyfikatora Bayesa, szczególnie w filtracji spamu, analizie sentymentu oraz kategoryzacji treści.

W ramach laboratorium przeprowadzono kompletny proces analizy danych tekstowych, obejmujący: tłumaczenie wielojęzycznych postów, tokenizację, usuwanie słów przystankowych, wektoryzację, budowę modelu oraz ewaluację jego skuteczności.

Rozdział I. Podstawy teoretyczne

I.1. Naiwny klasyfikator Bayesa

Naiwny klasyfikator Bayesa jest probabilistyczną metodą klasyfikacji opartą na twierdzeniu Bayesa z założeniem niezależności warunkowej cech. Zasada działania klasyfikatora opiera się o rozwiązanie zadania optymalizacyjnego w postaci:

$$C^* = arg \max_{\{C_i:\, i \geq 1\}} \left(Pr\{C_i\} \prod_{k=1}^{n} Pr\{x_k | C_i\} \right) \tag{1}$$

, gdzie

- $Pr\{C_i\}$ – prawdopodobieństwo wystąpienia klasy i, które można estymować poprzez iloczyn liczby wystąpień danej klasy do liczność zbioru testowego,
- $Pr\{x_k | C_i\}$ – prawdopodobieństwo warunkowe wystąpienia atrybutu x_k pod warunkiem wystąpienia klasy C_i.

Dla każdego atrybutu prawdopodobieństwo $Pr\{x_k | C_i\}$ można estymować w oparciu o zbiór uczący Z, wykorzystując wzór:

$$Pr\{x_k | C_i\} = \frac{\left| C_i^{x_k} \right|}{|C_i|} \tag{2}$$

, gdzie

- $\left| C_i^{x_k} \right|$ – oznacza liczbę „przykładów” (w zbiorze Z) z klasy C_i, dla których atrybut o numerze $k = 1, \ldots, n$ przyjmuje wartość x_k.

Kluczowym uproszczeniem jest założenie o warunkowej niezależności atrybutów:

$$\Pr\{\mathrm{X} | C_i\} = \Pr\{x_1, x_2, \ldots, x_\mathrm{n} | C_i\} = \prod_{k=1}^{n} Pr\{x_k | C_i\} \tag{3}$$

To "naiwne" założenie pozwala na znaczną redukcję złożoności obliczeniowej, choć w praktyce atrybuty często są skorelowane. Pomimo tego teoretycznego ograniczenia, naiwny klasyfikator Bayesa osiąga dobre wyniki w rzeczywistych zastosowaniach. [1]

I.2. Przetwarzanie języka naturalnego w klasyfikacji tekstu

Klasyfikacja dokumentów tekstowych wymaga przekształcenia tekstu na reprezentację numeryczną, którą może przetworzyć algorytm uczenia maszynowego. Proces ten obejmuje kilka kluczowych etapów [2]:

1. **Tokenizacja** – podział tekstu na jednostki leksykalne (tokeny), zazwyczaj słowa lub fragmenty słów. W tym procesie usuwa się znaki interpunkcyjne oraz normalizuje wielkość liter.
2. **Usuwanie słów przystankowych** – eliminacja słów o niskiej zawartości leksykalnej, które występują powszechnie w języku, ale nie wnoszą istotnej informacji dyskryminującej klasy. W języku angielskim są to słowa takie jak "the", "a", "an", "because", "instead".
3. **Stemming** – redukcja słów do ich rdzenia poprzez usunięcie końcówek fleksyjnych. Przykładowo, słowa "running", "runs", "ran" redukowane są do wspólnego rdzenia "run". Proces ten zwiększa uogólnienie modelu poprzez traktowanie różnych form tego samego słowa jako jednego atrybutu.
4. **Wektoryzacja** – przekształcenie tokensów na reprezentację numeryczną. Najpopularniejszą metodą jest model "bag of words" [12], w którym dokument reprezentowany jest jako wektor częstości występowania poszczególnych słów w słowniku.

W kontekście naiwnego klasyfikatora Bayesa, każdy unikalny token staje się cechą (atrybutem), a jego obecność lub częstość występowania w dokumencie służy do obliczenia prawdopodobieństw warunkowych $\Pr\{x_k|C_i\}$.

Rozdział II. Opis problemu

II.1. Treść zadania

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 gramatycznych 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).”

II.2. Opis problemu badawczego

Zebrano dwa zbiory danych:

- **Zbiór "Mandrill"** – posty odnoszące się do aplikacji Mandrill, zawierające dyskusje o problemach technicznych, konfiguracji, integracjach oraz doświadczeniach użytkowników z tym narzędziem
- **Zbiór "Inne"** – posty odnoszące się do małpy mandrill (*z ang. Mandryl barwnolicy*), gier komputerowych o tej nazwie, muzyki oraz innych kontekstów niezwiązanych z aplikacją

Dane charakteryzują się następującymi wyzwaniami:

1. **Wielojęzyczność** – posty napisane są w różnych językach, co wymaga tłumaczenia na język angielski przed procesowaniem.
2. **Różnorodność kontekstów** – nawet w ramach kategorii "Mandrill" występują różne konteksty: problemy techniczne, zapytania o funkcjonalność, porównania z konkurencją, co zwiększa wariancję cech w obrębie klasy.
3. **Ambiwalencja** – niektóre posty mogą zawierać elementy obu kategorii, np. "At first I thought mandrill was a tool, but it turned out to be an animal" lub "Mandrill appeared in both my biology and programming notes".
4. **Szum informacyjny** – posty z mediów społecznościowych zawierają nietypową ortografię, skróty, emotikony oraz linki, które wymagają odpowiedniego preprocessingu.

Problem stanowi realistyczne wyzwanie klasyfikacyjne, w którym skuteczność modelu zależy od jakości preprocessingu tekstu oraz zdolności klasyfikatora do identyfikacji słów kluczowych charakterystycznych dla każdej kategorii.

Rozdział III. Implementacja rozwiązania

III.1. Przygotowanie danych

Dane źródłowe zostały dostarczone w dwóch plikach CSV zawierających posty z serwisu Twitter. Ze względu na wielojęzyczność postów, pierwszym krokiem było ich tłumaczenie na język angielski przy użyciu narzędzia *LibreTranslate*.[4]

Kod. 1. Tłumaczenie postów z wykorzystaniem *LibreTranslate*

```
import requests
from json import loads
import csv

URL = "http://localhost:5000/translate"

with open('Lab-3-Zadanie-3-Dane Mandrill.csv', 'r') as f:
    reader = csv.reader(f)
    mandrill_data = list(x for x in reader if x != [])
    mandrill_data.pop(0)

with open('Lab-3-Zadanie-3-Dane Mandrill Przetłumaczone.csv', 'w')
as f:
    f.write("Post\n")

for line in mandrill_data:
    body = {
        "q": line,
        "source": "auto",
        "target": "en",
        "format": "text",
        "alternatives": 3,
        "api_key": ""
    }
    r = requests.post(URL, json=body)
    j = loads(r.text)
    translated_text = j['translatedText']
    with open('Lab-3-Zadanie-3-Dane Mandrill Przetłumaczone.csv',
'a') as f:
        f.write(" ".join(translated_text)+'\n')
```

Tłumaczenie przeprowadzono dla obu zbiorów danych, uzyskując przetłumaczone wersje postów zapisane w plikach o rozszerzeniu "* Przetłumaczone.csv". Proces tłumaczenia był niezbędny, ponieważ dalsze etapy preprocessingu (usuwanie słów przystankowych, stemming) wymagają jednolitego języka.

Po przetłumaczeniu dane zostały wczytane i połączone w jedną strukturę *DataFrame* biblioteki *pandas*.

Kod. 2. Wczytanie przetłumaczonych danych

```
import csv
import pandas as pd

with open('Lab-3-Zadanie-3-Dane Mandrill Przetłumaczone.csv', 'r')
as f:
    reader = csv.reader(f)
    mandrill_data = list(x[0] for x in reader if x != [])
    mandrill_data.pop(0)

with open('Lab-3-Zadanie-3-Dane Inne Przetłumaczone.csv', 'r') as
f:
    reader = csv.reader(f)
    other_data = list(x[0] for x in reader if x != [])
    other_data.pop(0)
```

III.2. Tokenizacja i preprocessing tekstu

Kluczowym etapem przygotowania danych tekstowych jest ich przekształcenie na reprezentację, którą może przetworzyć model klasyfikacyjny. Zaimplementowano funkcję `tokenize()` realizującą kompletny preprocessing pojedynczego tekstu. Funkcja korzysta z biblioteki *nltk*, będącej zbiorem narzędzi do przetwarzania języka naturalnego. [3]

Kod. 3. Funkcja implementująca algorytm naiwnego klasyfikatora Bayesa

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.stem import PorterStemmer

nltk.download('stopwords')
nltk.download('words')
porter = PorterStemmer()

def tokenize(text: str) -> list:
    # Usunięcie linków, tagów @
    text = " ".join([x for i, x in enumerate(text.split(' '))
                     if not x.startswith('http')
                     and not (i == 0 and x.startswith('@'))])

    # Tokenyzacja z usunięciem interpunkcji
    tokenizer = RegexpTokenizer(r'[A-Za-z_\']+')
    tokens = tokenizer.tokenize(text.lower())

    # Usunięcie wyrazów przystankowych
    stop_words = set(stopwords.words('english'))
    filtered_tokens = set([porter.stem(word) for word in tokens
                           if word not in stop_words and len(word)
>= 2])
    return filtered_tokens
```

Funkcja `tokenize()` realizuje następujące operacje:
1. **Usuwanie szumu** – eliminacja linków URL oraz tagów użytkowników (@username) z początku postów, które nie wnoszą informacji dyskryminującej klasy.
2. **Tokenizacja** – podział tekstu na słowa z wykorzystaniem wyrażenia regularnego akceptującego tylko litery alfabetu oraz apostrofy (dla kontakcji jak "don't", "it's").
3. **Normalizacja** – konwersja wszystkich liter na małe, aby traktować "Mandrill" i "mandrill" jako ten sam token.
4. **Filtracja słów przystankowych** – usunięcie 179 najczęstszych słów angielskich (jak "the", "a", "is", "are") z wykorzystaniem korpusu NLTK.
5. **Stemming**- redukcja słów do rdzenia z wykorzystaniem algorytmu Portera, np. "emails" → "email", "sending" "send".
6. **Filtracja długości** – usunięcie tokenów krótszych niż 2 znaki.

Funkcja zwraca zbiór (set) unikalnych tokenów, co automatycznie eliminuje duplikaty w obrębie jednego dokumentu.
Do przetworzenia wszystkich postów zaimplementowano funkcję pomocniczą:

Kod. 4. Przetwarzanie kolekcji tekstów

```
def tokenize_lines(lines_of_text: list[str]) -> tuple[list, set]:
    text_tokens = []
    unique_tokens = set()
    for line in lines_of_text:
        filtered_tokens = tokenize(line)
        text_tokens.append(filtered_tokens)
        unique_tokens.update(filtered_tokens)
    return text_tokens, unique_tokens
```

Funkcja przetwarza wszystkie posty zwracając listę zestawów tokenów dla każdego dokumentu oraz zbiór wszystkich unikalnych tokenów występujących w korpusie.

III.3. Wektoryzacja danych

Po tokenizacji przeprowadzono konsolidację danych w strukturze *DataFrame* oraz podział na zbiory treningowy i testowy, z użyciem modułu *model_selection* z biblioteki *scikit-learn*.

Kod. 5. Przygotowanie zbiorów danych

```
from sklearn.model_selection import train_test_split

mandrill_tokens, unique_tokens = tokenize_lines(mandrill_data)
other_tokens, temp = tokenize_lines(other_data)
unique_tokens.update(temp)

data = pd.DataFrame({'clean_text': [], 'text': [], 'label': []})
mandrill_df = pd.DataFrame({
    'text': mandrill_data[i],
    'clean_text': " ".join(tokens),
    'label': 'Mandrill'
} for i, tokens in enumerate(mandrill_tokens))

other_df = pd.DataFrame({
    'text': other_data[i],
    'clean_text': " ".join(tokens),
    'label': 'Inne'
} for i, tokens in enumerate(other_tokens))

data = pd.concat([mandrill_df, other_df])

X = data["clean_text"]
y = data["label"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, stratify=y
)
```

Zbiór danych został podzielony w stosunku 60:40 na zbiór treningowy i testowy z zachowaniem proporcji klas (stratyfikacja). Łącznie zebrano 299 postów.
Wektoryzację tekstów przeprowadzono metodą bag-of-words z wykorzystaniem `CountVectorizer` z biblioteki *scikit-learn* [5]:

Kod. 6. Funkcje implementujące dwie metryki odległości - euklidesowa i Manhattan

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

CountVectorizer tworzy słownik wszystkich unikalnych tokenów występujących w zbiorze treningowym i reprezentuje każdy dokument jako wektor częstości występowania tych tokenów. Metoda `fit_transform()` buduje słownik na zbiorze treningowym, natomiast `transform()` stosuje ten sam słownik do zbioru testowego. [5]

III.4. Budowa modelu klasyfikacyjnego

Do klasyfikacji wykorzystano implementację naiwnego klasyfikatora Bayesa dla danych o rozkładzie wielomianowym (*MultinomialNB*) z biblioteki *scikit-learn* [6]

Kod. 7. Trening modelu klasyfikacyjnego

```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(X_train_vec, y_train)
```

MultinomialNB jest odpowiednią wersją naiwnego klasyfikatora Bayesa dla danych reprezentujących częstości (liczby wystąpień), jakie generuje *CountVectorizer*. Model zakłada rozkład wielomianowy cech w obrębie każdej klasy. [6]
Proces uczenia polega na estymacji prawdopodobieństw:

- $\Pr\{C_i\}$ – prawdopodobieństwa *a priori* dla każdej klasy na podstawie częstości występowania w zbiorze treningowym
- $\Pr\{token_k | C_i\}$ – prawdopodobieństwa warunkowego wystąpienia każdego tokenu w dokumentach danej klasy

Implementacja *scikit-learn* automatycznie stosuje wygładzanie Laplace'a (domyślnie $\alpha = 1$) [7], które zapobiega problemowi zerowych prawdopodobieństw dla tokenów nieobserwowanych w zbiorze treningowym.

Rozdział IV. Wyniki eksperymentów

IV.1. Wyniki klasyfikacji na zbiorze testowym

Po wytrenowaniu modelu przeprowadzono ewaluację na zbiorze testowym składającym się ze 120 postów (60 z każdej klasy).

Kod. 8. Ewaluacja modelu

```
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score

y_pred = model.predict(X_test_vec)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nMacierz pomyłek:")
print(confusion_matrix(y_test, y_pred))
print("\nRaport klasyfikacji:")
print(classification_report(y_test, y_pred))
```

Model osiągnął następujące wyniki:
Dokładność (Accuracy): 0.85 (85%)

Tab. 1. Macierz pomyłek klasyfikacji

	Predykcja: Inne	Predykcja: Mandrill
Rzeczywistość: Inne	**49**	11
Rzeczywistość: Mandrill	7	**53**

Tab. 2. Zestawienie wybranych metryk jakości klasyfikacji

Klasa	Precision	Recall
Inne	0.93	0.72
Mandrill	0.77	0.95

Uzyskane wyniki wskazują na asymetryczne charakterystyki klasyfikacji w odniesieniu do obu klas decyzyjnych. Precyzja (ang. *precision*) wyniosła 0.93 dla klasy „Inne" oraz 0.77 dla klasy „Mandrill". Oznacza to, że prawdopodobieństwo poprawnej klasyfikacji przy predykcji klasy „Inne" jest istotnie wyższe niż w przypadku predykcji klasy „Mandrill". Innymi słowy, gdy klasyfikator wskazuje na przynależność obiektu do klasy „Inne", można to przyjąć z wysoką pewnością, natomiast predykcja klasy „Mandrill" obarczona jest większym ryzykiem błędu. [9]

Analiza czułości (ang. *recall*) przedstawia odwrotną zależność. Dla klasy „Inne" czułość wynosi 0.72, podczas gdy dla klasy „Mandrill" osiąga wartość 0.95. Świadczy to o wysokiej skuteczności modelu w identyfikacji postów odnoszących się do aplikacji Mandrill – klasyfikator poprawnie rozpoznaje 95% przypadków należących do tej kategorii. Jednocześnie w przypadku klasy „Inne" zaobserwowano niższą czułość, co

oznacza, że 28% obiektów z tej kategorii zostaje błędnie przypisanych do klasy „Mandrill".[9]

IV.2. Analiza błędów klasyfikacji

Szczegółowa analiza macierzy pomyłek pozwala na identyfikację charakterystycznych wzorców błędów popełnianych przez wyuczony klasyfikator oraz zrozumienie mechanizmów prowadzących do niepoprawnych decyzji. [10]

IV.2.1. Charakterystyka błędów typu I i II

Błędy fałszywie pozytywne dla klasy „Mandrill": 17 przypadków

Stanowią one obiekty z kategorii „Inne" błędnie zaklasyfikowane jako „Mandrill". Jest to dominujący typ błędu w badanym modelu, odpowiadający za 85% wszystkich niepoprawnych klasyfikacji. Analiza tego zjawiska wskazuje na tendencję klasyfikatora do nadmiernego przypisywania obiektów do kategorii „Mandrill".

Prawdopodobnym źródłem tego typu błędów jest występowanie w postach o tematyce niezwiązanej z aplikacją słownictwa, które w zbiorze treningowym było silnie skorelowane z klasą „Mandrill". Przykładowo, słowa takie jak „behavior", „observation" czy „study", charakterystyczne dla kontekstu zoologicznego, mogą również występować w dyskusjach technicznych dotyczących zachowania aplikacji czy obserwacji jej działania.

Błędy fałszywie negatywne dla klasy „Mandrill": 3 przypadki

Reprezentują obiekty rzeczywiście odnoszące się do aplikacji Mandrill, które zostały błędnie zaklasyfikowane jako „Inne". Ten typ błędu występuje znacznie rzadziej, co przekłada się na bardzo wysoką wartość czułości (0.95) dla klasy „Mandrill". Model wykazuje zatem wysoką skuteczność w rozpoznawaniu charakterystycznego słownictwa technicznego związanego z aplikacją do wysyłki wiadomości elektronicznych.

IV.2.2. Asymetria w rozkładzie błędów

Stosunek liczebności błędów fałszywie pozytywnych do fałszywie negatywnych wynosi $17:3 \approx 5.7:1$, co ujawnia wyraźną asymetrię w zachowaniu klasyfikatora. Model demonstruje silną tendencję do przypisywania przypadków granicznych do klasy „Mandrill". Zjawisko to może być wyjaśnione następującymi czynnikami:

1. **Specyficzność słownictwa domenowego**: Terminy związane z aplikacją Mandrill, takie jak „email", „API", „webhook", „SMTP" czy „server", charakteryzują się wysoką swoistością i są łatwo rozpoznawalne przez model bayesowski. Obecność nawet niewielkiej liczby takich tokenów w dokumencie prowadzi do wysokich prawdopodobieństw warunkowych

Pr $\{token_k|C_{Mandrill}\}$, co w konsekwencji faworyzuje klasyfikację do tej kategorii.

2. **Różnorodność klasy „Inne"**: Kategoria „Inne" obejmuje szeroki zakres kontekstów semantycznych, od opisów zoologicznych, przez gry komputerowe, po muzykę. Ta różnorodność utrudnia wyuczenie się jednolitego wzorca probabilistycznego dla tej klasy, prowadząc do większego rozproszenia prawdopodobieństw warunkowych dla poszczególnych tokenów.

IV.3. Testy na danych syntetycznych

W celu głębszej weryfikacji zdolności generalizacji modelu przeprowadzono testy na 40 syntetycznie wygenerowanych postach przez model językowy, które imitują rzeczywiste posty z obu kategorii.

Kod. 9. Testowanie na danych syntetycznych

```
test_posts = [
    ("Email delivery stopped right after we changed the mandrill settings", "Mandrill"),
    ("Not sure if mandrill supports this kind of webhook event", "Mandrill"),
    ("Our backend logs show mandrill rejecting messages again", "Mandrill"),
# (...) pozostałe posty
    ("Reading about mandrill ecology was surprisingly interesting", "Inne"),
    ("Is Mandrill down right now?", "Mandrill")
]

synthetic_data = pd.DataFrame(
    {'text': post[0], 'clean_text': " ".join(tokenize(post[0])), 'label': post[1]} for post in test_posts
)
synthetic_x_test_vec = vectorizer.transform(synthetic_data['clean_text'])
synthetic_y_test = synthetic_data['label']
synthetic_y_pred = model.predict(synthetic_x_test_vec)

print("Accuracy:", accuracy_score(synthetic_y_test, synthetic_y_pred))
print("\nMacierz pomyłek:")
print(confusion_matrix(synthetic_y_test, synthetic_y_pred, labels=['Inne', 'Mandrill']))

synthetic_data_precision = (precision_score(synthetic_y_test, synthetic_y_pred, pos_label='Inne'), precision_score(synthetic_y_test, synthetic_y_pred, pos_label='Mandrill'))
synthetic_data_recall = (recall_score(synthetic_y_test, synthetic_y_pred, pos_label='Inne'), recall_score(synthetic_y_test, synthetic_y_pred, pos_label='Mandrill'))

print( "|           |   Inne   | Mandrill |")
print( "|===========+==========+==========|")
print(f"| Precision |   {synthetic_data_precision[0]:4.2f}   |
```

```
34          {synthetic_data_precision[1]:4.2f}   |")
35          print(f"| Recall    |   {synthetic_data_recall[0]:4.2f}   |
36          {synthetic_data_recall[1]:4.2f}   |")
```

IV.3.1. Wyniki na danych syntetycznych

- Skuteczność: 77.5%

Tab. 3. Macierz pomyłek klasyfikacji dla danych syntetycznych

	Predykcja: Inne	Predykcja: Mandrill
Rzeczywistość: Inne	**15**	5
Rzeczywistość: Mandrill	4	**16**

Tab. 4. Zestawienie wybranych metryk jakości klasyfikacji dla danych syntetycznych

Klasa	Precision	Recall
Inne	0.79	0.76
Mandrill	0.75	0.80

IV.3.2. Przykłady poprawnej klasyfikacji

1. "Email delivery stopped right after we changed the mandrill settings" – **Mandrill**
2. "Spent the whole night debugging mandrill responses from the API" – **Mandrill**
3. "The mandrill in the enclosure barely moved all afternoon" – **Inne**
4. "Children were fascinated by the mandrill's colorful face" – **Inne**

IV.3.3. Przykłady błędnej klasyfikacji

1. "Our backend logs show mandrill rejecting messages again" – sklasyfikowano jako **Inne** (powinno być Mandrill)
2. "That mandrill looked surprisingly calm compared to others" – sklasyfikowano jako **Mandrill** (powinno być Inne)
3. "Mandrill documentation is not always up to date" – sklasyfikowano jako **Inne** (powinno być Mandrill)

Analiza błędów na danych syntetycznych ujawnia następujące wzorce:

1. **Kontekst techniczny** – Słowa takie jak "logs", "backend", "rejecting", "messages" są silnie skorelowane z kategorią Mandrill w zbiorze treningowym. Gdy występują razem ze słowem "mandrill", powinny sugerować kontekst aplikacji, jednak w przypadku pierwszego przykładu błędu model nie rozpoznał tego kontekstu.
2. **Ambiwalentne słowa** - Słowa takie jak "looked", "calm", "compared" są charakterystyczne dla opisów zwierząt, jednak w drugim przykładzie błędu model został wprowadzony w błąd przez inne tokeny w zdaniu.

Mimo tych ograniczeń, skuteczność 77.5% na danych syntetycznych, które celowo zawierały trudne przypadki graniczne, potwierdza dobrą zdolność generalizacji modelu.

Podsumowanie

W ramach niniejszego laboratorium przeprowadzono kompletny proces klasyfikacji postów z serwisu Twitter przy użyciu naiwnego klasyfikatora Bayesa. Zadanie polegało na automatycznym rozróżnianiu postów odnoszących się do aplikacji Mandrill od postów dotyczących innych znaczeń tego słowa.

Główne wnioski:

1. **Skuteczność naiwnego klasyfikatora Bayesa** – Pomimo upraszczającego założenia o niezależności tokenów, model osiągnął zadowalającą skuteczność w praktycznym zastosowaniu do klasyfikacji tekstu.
2. **Znaczenie preprocessingu** – Jakość tokenizacji, usuwania słów przystankowych oraz stemmingu ma kluczowy wpływ na wyniki klasyfikacji. Dobór odpowiednich operacji preprocessing jest równie ważny jak wybór algorytmu.
3. **Ograniczenia** – Głównym źródłem błędów są posty ambiwalentne lub zawierające niewiele charakterystycznych słów kluczowych. Dalsze usprawnienia mogłyby obejmować wykorzystanie n-gramów (bigramów, trigramów) zamiast pojedynczych słów. [11]
4. **Skalowalność** – Naiwny klasyfikator Bayesa charakteryzuje się niską złożonością obliczeniową zarówno w fazie uczenia, jak i predykcji, co czyni go odpowiednim do zastosowań wymagających przetwarzania dużych wolumenów danych.

Przeprowadzone laboratorium wykazało praktyczną przydatność naiwnego klasyfikatora Bayesa w zadaniach przetwarzania języka naturalnego oraz udowodniło, że przy odpowiednim preprocessingu tekstu można osiągnąć wysoką skuteczność klasyfikacji nawet stosując stosunkowo prosty model probabilistyczny.

Bibliografia

[1] Hoffmann R.: „Metody eksploracji danych - Wykład 5. Klasyfikacja Naive Bayes slajdy”, Materiały dydaktyczne WAT, 2025.
[2] https://www.geeksforgeeks.org/python/nltk-nlp/ [dostęp 22.12.2025]
[3] https://www.nltk.org/index.html [dostęp 22.12.2025]
[4] https://docs.libretranslate.com/ [dostęp 22.12.2025]
[5] https://scikit-learn.org/stable/modules/feature_extraction.html [dostęp 22.12.2025]
[6] https://scikit-learn.org/stable/modules/naive_bayes.html [dostęp 22.12.2025]
[7] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB [dostęp 22.12.2025]
[8] https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics [dostęp 22.12.2025]
[9] https://en.wikipedia.org/wiki/Precision_and_recall [dostęp 06.01.2026]
[10] https://en.wikipedia.org/wiki/Confusion_matrix [dostęp 06.01.2026]
[11] https://en.wikipedia.org/wiki/Word_n-gram_language_model [dostęp 07.01.2026]
[12] https://en.wikipedia.org/wiki/Bag-of-words_model [dostęp 07.01.2026]

Spis tabel

Załączniki

1. Plik notebook *Lab-3-Zadanie-3-Obliczenia.ipynb* – kompletny kod przeprowadzonych eksperymentów obejmujący:
    a. Proces wstępnej obróbki danych z użyciem biblioteki *nltk*
    b. Podział danych na treningowe i testowe
    c. Wektoryzacje danych z użyciem biblioteki *scikit-learn*
    d. Utworzenie i naukę modelu wielomianowego naiwnego klasyfikatora Bayesa
    e. Ocenę klasyfikacji modelu
2. Plik notebook *Lab-3-Zadanie-3-Tłumaczenie-wpisów.ipynb* – kompletny kod procesu tłumaczenia wpisów z użyciem narzędzia *LibreTranslate*
3. Plik danych *Lab-3-Zadanie-3-Dane-Mandrill.csv* – oryginalne wpisy dot. aplikacji Mandrill
4. Plik danych *Lab-3-Zadanie-3-Dane-Inne.csv* – oryginalne wpisy dot. innych tematów niż aplikacja Mandrill
5. Plik danych *Lab-3-Zadanie-3-Dane-Mandrill-Przetłumaczone.csv* – przetłumaczone na język angielski wpisy dot. aplikacji Mandrill
6. Plik danych *Lab-3-Zadanie-3-Dane-Inne-Przetłumaczone.csv* – przetłumaczone na język angielski wpisy dot. innych tematów niż aplikacja Mandrill