SPRAWOZDANIE

Z ĆWICZENIA LABORATORYJNEGO NR 3

ZADANIE 2

**Klimiuk Maciej, Mika Hanna**

**WCY22IJ1S1**

# Klasyfikacja postów na podstawie naiwnego klasyfikatora bayesowskiego

Marndill

# Treść zadania

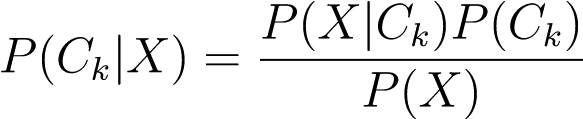
Zadanie 3 Zadanie odnosi się do praktyki całkiem częstego zastosowania naiwnego klasyfikatora bayesowskiego do klasyfikacji dokumentów. Przykładem zastosowania tego klasyfikatora jest klasyfikacja wiadomości e-mail jako nas interesującą lub spam. Inne przykłady. Odpowiadamy na pytanie czy dany post wyraża 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).”

# Wprowadzenie

Zadanie polega na zastosowaniu naiwnego klasyfikatora Bayesa do klasyfikacji postów opublikowanych w serwisie Twitter, które odnoszą się do aplikacji *Mandrill*. Klasyfikator ma rozróżnić posty dotyczące aplikacji *Mandrill* od postów na inne tematy związane z rzeczownikiem *mandrill*. Dane do analizy zostały przedstawione w wielojęzycznym formacie.

# Opis teoretyczny

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

*,* (1)

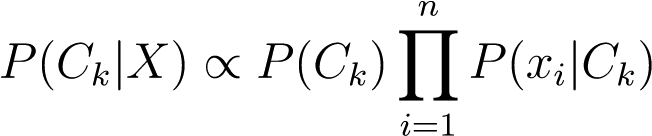
gdzie:

* *P*(*Ck*|*X*) – prawdopodobieństwo, że obserwacja *X* należy do klasy *Ck*

(a posteriori),

* *P*(*X*|*Ck*) – prawdopodobieństwo zaobserwowania *X* przy założeniu, że należy do klasy *Ck*,
* *P*(*Ck*) – prawdopodobieństwo a priori wystąpienia klasy *Ck*,
* *P*(*X*) – całkowite prawdopodobieństwo obserwacji *X*.

W przypadku naiwnego klasyfikatora Bayesa zakłada się niezależność cech, co pozwala uprościć obliczenia:

*,* (2)

gdzie *xi* to poszczególne cechy obserwacji *X*.

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:

1. **Precision (precyzja)**: wskazuje, jaki procent obserwacji sklasyfikowanych jako pozytywne rzeczywiście jest pozytywny. Precyzja jest określona wzorem:

Precision = Obraz zawierający czarne, ciemność

Opis wygenerowany automatycznie*.* (3)

1

1. **Recall (czułość)**: mierzy zdolność modelu do wykrywania wszystkich rzeczywistych pozytywnych przypadków. Czułość definiuje się jako:

Recall = Obraz zawierający czarne, ciemność

Opis wygenerowany automatycznie*.* (4)

1. **F1-score**: jest średnia harmoniczna precyzji i czułości, co pozwala na ich jednoczesne uwzględnienie. Wzór na *F1-score*:

Precision · Recall

F1-score = 2 · *.* (5)

Precision + Recall

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.

# Przygotowanie danych

Do przetworzenia tekstu zastosowano następujące kroki:

1. Transkrypcja postów na język Angielski
2. Usuniecie słów przystankowych (ang. *stop words*) w jezyku angielskim.
3. Tokenizacja tekstu – podziału na leksemy.
4. Normalizacja tekstu, w tym sprowadzenie do małych liter i usuniecie znaków interpunkcyjnych.
5. Zmiana słów na wektory

vectorizer = TfidfVectorizer(stop\_words='english', max\_features=5000)

# Metodyka

Model został zaimplementowany w języku Python, wykorzystując biblioteki pandas i scikit-learn. Głowne kroki obejmowały:

* Przygotowanie zbioru danych do uczenia modelu.
* Podział danych na zbiór treningowy i testowy.
* Zmiana słów na wektory:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* Trenowanie klasyfikatora naiwnego Bayesa.
* Walidacja modelu za pomocą miar, takich jak dokładność (ang. *accuracy*).

# Wyniki

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*.

## Wyniki klasyfikacji

Poniżej przedstawiono wyniki klasyfikacji dla dwóch klas: *Mandrill* i *inne*, wraz z podsumowaniem:

|  |  |  |  |
| --- | --- | --- | --- |
| 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.

## Analiza macierzy konfuzji

Macierz konfuzji przedstawia wyniki klasyfikacji modelu dla dwóch klas: **Mandrill** i **inne**. Elementy macierzy zawierają liczby obserwacji poprawnie i błędnie sklasyfikowanych dla każdej klasy.

Obraz zawierający zrzut ekranu, tekst, kwadrat, diagram

Opis wygenerowany automatycznie Figure 1: Macierz konfuzji modelu klasyfikacji.

### Miary jakości klasyfikacji

Na podstawie macierzy konfuzji obliczono następujące wskaźniki jakości klasyfikacji:Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

### Wnioski

Model klasyfikacyjny osiągnął wysokie wyniki we wszystkich miarach, co oznacza, że jest skuteczny w rozróżnianiu miedzy klasami **Mandrill** i **inne**. Wyniki precyzji, czułości i F1-score wynoszą 0.94, a dokładność modelu to 0.93.Krzywa ROC i wartość AUC

## Analiza Krzywej ROC

Krzywa ROC (Receiver Operating Characteristic) to narzędzie stosowane w ocenie wydajności klasyfikatorów binarnych. Jej celem jest wizualizacja zdolności modelu do rozróżniania między klasami pozytywną (np. "Mandrill") a negatywną (np. "inne") w różnych progach decyzyjnych.

Na poniższym wykresie przedstawiono krzywą ROC dla modelu klasyfikacyjnego. Wartość AUC (Area Under the Curve) wynosi 0.9774, co świadczy o bardzo wysokiej skuteczności klasyfikatora.

Obraz zawierający tekst, zrzut ekranu, diagram, Wykres

Opis wygenerowany automatycznie

Figure 2:Krzywa ROC

Składniki krzywej ROC:

1. True Positive Rate (TPR)
2. False Positive Rate (FPR)

### Wartość AUC (Area Under Curve):

AUC po weryfikacji: 0.9774193548387096

# Wnioski

Przeprowadzona analiza modelu klasyfikacyjnego pozwala na sformułowanie następujących wniosków: **Dokładność modelu**: Model osiągnął wysoką dokładność klasyfikacji, wynoszącą 93%. Jest to zadowalający wynik, który wskazuje na skuteczność zastosowanego podejścia w rozpoznawaniu klas "Mandrill" i "inne". **Miary klasyfikacyjne**: **Precyzja (Precision)**: Wartość precyzji dla obu klas wynosiła powyżej 0.93, co świadczy o małej liczbie fałszywych trafień (false positives). **Czułość (Recall)**: Czułość również utrzymywała się na poziomie 0.93, co oznacza, że model dobrze identyfikuje rzeczywiste przypadki (true positives). **F1-Score**: Uzyskane F1-score dla obu klas wyniosło 0.94 i 0.93, co wskazuje na równowagę między precyzją a czułością. **Macierz konfuzji**: Analiza macierzy konfuzji ujawnia, że: Klasa "Mandrill" została błędnie zaklasyfikowana w 2 przypadkach (z 31 obserwacji).Klasa "inne" została błędnie zaklasyfikowana w 2 przypadkach (z 30 obserwacji).  
Wyniki te wskazują na zrównoważony poziom błędów pomiędzy klasami. **Krzywa ROC i AUC**: Wartość AUC wynosząca 0.98 potwierdza wysoką jakość klasyfikatora. Krzywa ROC jest zbliżona do idealnej, co oznacza, że model skutecznie odróżnia klasy nawet przy zmieniającym się progu decyzyjnym. **Obserwacje ogólne**: Model radzi sobie bardzo dobrze z danymi testowymi, co wskazuje na jego zdolność generalizacji. Istnieje niewielka różnica w wynikach między klasami, co sugeruje, że dane są dobrze zbilansowane, a model nie jest uprzedzony wobec żadnej z klas. **Rekomendacje**: W przyszłości warto rozważyć zwiększenie zbioru danych w celu dalszej poprawy wyników. Można także eksperymentować z innymi algorytmami klasyfikacji lub metodami selekcji cech, aby zbadać potencjalne możliwości poprawy.

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()