Raport badawczy na temat 1000 największych kanałów na platformie Youtube w 2023 roku.

Gracjan Kościński

13 maja 2024

Spis treści

1 Baza danych oraz preprocessing

1.1 Baza danych

Korzystałem z bazy danych dostępnej na stronie kaggle, która zawiera 1000 kanałów z największą liczbą subskrypcji na Youtube w 2023 roku. Baza danych jest niekompletna, brakuje w niej wielu wartości, co pozwala sprawdzić poniższy kod:

```
print(df.isnull().sum())
# Output:
# rank                                  0
# Youtuber                              0
# subscribers                           0
# video views                           0
# category                             46
# Title                                 0
# uploads                               0
# Country                              25
# Abbreviation                         26
# channel_type                         30
# video_views_rank                      1
# country_rank                        116
# channel_type_rank                    33
# video_views_for_the_last_30_days     56
# lowest_monthly_earnings               0
# highest_monthly_earnings              0
# lowest_yearly_earnings                0
# highest_yearly_earnings               0
# subscribers_for_last_30_days        337
# created_year                          5
# created_month                         5
# created_date                          5
# Gross tertiary education enrollment (%)  123
# Population                           123
# Unemployment rate                    123
# Urban_population                     123
# Latitude                             123
# Longitude                            123
```

Jak widać nazewnictwo kolumn nie korzysta z jednej konwencji.

1.2 Preprocessing

Naprawę bazy danych postanowiłem rozpocząć od ujednolicenia nazw kolumn na "snake case".

```
# zmiana nazwy na nazwe bez spacji
df.rename(columns={"video views": "video_views",
 "Unemployment rate": "unemployment_rate"}, inplace=True)
df.rename(columns={"Gross tertiary education enrollment (%)":
 "gross_tertiary_education_enrollment"},inplace=True)
# zamiana, aby kazda kolumna zaczynala sie mala litera
df.columns = [col.lower() for col in df.columns]
```

Następnie postanowiłem połączyć kolumny dotyczące roku stworzenia kanału, miesiąca stworzenia kanału oraz dnia stworzenia kanału w jedną wspólną kolumnę opisującą datę stworzenia kanału.

```
df['created_year']=df['created_year'].astype(str).str.split('.').str[0]
df['created_month'] = df['created_month'].astype(str)
df['created_date'] = df['created_date'].astype(str).str.split('.').str[0]
df['created_at'] = df['created_date'] + '-' + df['created_month'] + '-' + df['created_year']
df['created_at'] = df['created_at'].replace("nan-nan-nan", np.nan)
df['created_at'] = pd.to_datetime(df['created_at'])
df.drop(['created_year', 'created_month', 'created_date'], axis=1, inplace=True)
```

Jako kolejny krok postanowiłem pozbyć się kolumn, które uznałem za mało istotne w dalszej analizie, tj: longitude, latitude oraz channel type rank. Kolejnym krokiem było pozbycie się sztucznych kanałów, na których nie ma żadnych treści, czyli takich gdzie liczba wrzuconych filmów lub liczba wyświetleń filmów jest równa 0. Jako kolejny krok postanowiłem pozbyć się nazw kanałów, które posiadają w swojej nazwie znaki spoza tablicy znaków ASCII.

```
df = df[~df['youtuber'].str.contains('ý')]
df = df[~df['youtuber'].str.contains('½')]
df = df[~df['youtuber'].str.contains('¿')]
df = df[~df['youtuber'].str.contains('ï')]
df = df[~df['youtuber'].str.contains('ż')]
df = df[~df['title'].str.contains('ý')]
df = df[~df['title'].str.contains('½')]
df = df[~df['title'].str.contains('¿')]
df = df[~df['title'].str.contains('ï')]
df = df[~df['title'].str.contains('ż')]
```

W kolejnym kroku postanowiłem usunąć te kanały, w których brakuje przypisanej kategorii, po tej operacji sprawdziłem również balans klas dla "category", ponieważ to ona będzie klasyfikowana.

```
df = df.dropna(subset=['category'])
class_counts = df['category'].value_counts()
print(class_counts)
# Output:
# category
# Entertainment            183
# Music                    172
# People & Blogs            82
# Gaming                    75
# Comedy                    59
# Education                 34
# Howto & Style             29
# Film & Animation          28
# News & Politics           24
# Science & Technology      14
# Shows                     10
# Sports                     9
# Pets & Animals             3
# Movies                     2
# Nonprofits & Activism      2
# Autos & Vehicles           2
# Trailers                   2
# Name: count, dtype: int64
```

Jak widać niektóre kategorie są zbyt mało liczne, aby ich analiza miała sens więc należy je usunąć.

```
df = df[~df['category'].str.contains('Sports')]
df = df[~df['category'].str.contains('Trailers')]
df = df[~df['category'].str.contains('Pets & Animals')]
df = df[~df['category'].str.contains('Movies')]
df = df[~df['category'].str.contains('Nonprofits & Activism')]
df = df[~df['category'].str.contains('Autos & Vehicles')]
df = df[~df['category'].str.contains('Shows')]
df = df[~df['category'].str.contains('Science & Technology')]
```

Następnie pozostało wypełnić kolumny 'subscribers_for_last_30_ days' oraz 'video_views_for_last_30_days'. Do tego celu wykorzystałem regresję liniową tak jak jest to pokazane w kodzie poniżej

```
# użycie LinearRegression do przewidzenia wartości dla 'subscribers_for_last_30_days'
# oraz 'video_views_for_the_last_30_days'
label_encoder = LabelEncoder()

df['encoded_subscribers'] = label_encoder.fit_transform(df['subscribers'])
df['encoded_video_views'] = label_encoder.fit_transform(df['video_views'])

features = ['encoded_subscribers', 'encoded_video_views']
target = 'subscribers_for_last_30_days'

linear_regression = LinearRegression()
train_df = df.dropna(subset=[target])
test_df = df[df[target].isnull()]

linear_regression.fit(train_df[features], train_df[target])
predictions = linear_regression.predict(test_df[features])

df.loc[df[target].isnull(), target] = predictions

features = ['encoded_subscribers', 'encoded_video_views']
target = 'video_views_for_the_last_30_days'

linear_regression = LinearRegression()
train_df = df.dropna(subset=[target])
test_df = df[df[target].isnull()]

linear_regression.fit(train_df[features], train_df[target])
predictions = linear_regression.predict(test_df[features])

df.loc[df[target].isnull(), target] = predictions
```

Ostatnim działaniem w ramach preprocessingu pozostało już tylko wpisane danych o pewnych krajach. Dane te pojawiają się w bazie, ale nie są wpisane dla każdego wystąpienia kraju. Na przykładzie Kanady:

```
df.loc[df['country'] == 'Canada', 'gross_tertiary_education_enrollment'] = 88.2
df.loc[df['country'] == 'Canada', 'population'] = 36991981.0
df.loc[df['country'] == 'Canada', 'unemployment_rate'] = 5.56
df.loc[df['country'] == 'Canada', 'urban_population'] = 30628482.0
```

Po tych wszystkich operacjach wreszcie nasza baza danych jest gotowa do klasyfikacji.

```
print(df.isnull().sum())
# rank                                  0
# youtuber                              0
# subscribers                           0
# video_views                           0
# category                              0
# title                                 0
# uploads                               0
# country                               0
# abbreviation                          0
# channel_type                          0
# video_views_rank                      0
# country_rank                          0
# video_views_for_the_last_30_days      0
# lowest_monthly_earnings               0
# highest_monthly_earnings              0
# lowest_yearly_earnings                0
# highest_yearly_earnings               0
# subscribers_for_last_30_days          0
# gross_tertiary_education_enrollment   0
# population                            0
# unemployment_rate                     0
# urban_population                      0
# created_at                            0
# dtype: int64
```

2 Klasyfikacja

2.1 Przygotowanie danych do klasyfikacji

Przed użyciem klasyfikatorów należy odpowiednio przygotować dane. "Category"oraz "channel_type", to dane tekstowe, natomiast wszystkie powinny być numeryczne, więc zostają one przerobione za pomocą Label encodera, który każdej kategorii przyporządkowuje liczbę.

```
le = LabelEncoder()
df['category'] = le.fit_transform(df['category'])
df['channel_type'] = le.fit_transform(df['channel_type'])
```

Przykładowo dla "category", wynikiem tego działania jest takie przyporządkowanie:

0 - Comedy

1 - Education

2 - Entertainment

3 - Film & Animation

4 - Gaming

5 - Howto & Style

6 - Music

7 - News & Politics

8 - People & Blogs

Następnie wybrałem interesujące mnie kolumny wejściowe, jak i kolumnę której klasyfikacji chcę dokonać. Dane wejściowe należało również przeskalować za pomocą Standard Scaller(StandardScaler służy do przeskalowania danych wejściowych w taki sposób, aby miały średnią równą zero i wariancję równą jeden), ponieważ, jeśli cechy mają różne skale, a jedna z cech ma znacznie większy zakres wartości niż pozostałe, może to spowodować, że model będzie traktować tę cechę jako istotniejszą. Skalowanie danych zapobiega takiej dominacji, co prowadzi do bardziej stabilnego i dokładnego modelu.Dane podzielono na dwa zbiory. Zbiór treningowy (75%) oraz zbiór testowy (25%).

```
# Wybór cech i klasy docelowej
X = df[['uploads', 'video_views', 'video_views_for_the_last_30_days',
'subscribers','subscribers_for_last_30_days','channel_type']]
y = df['category']
X_scaled = StandardScaler().fit_transform(X)

# Podział na zestawy treningowe i testowe
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25,
random_state=42)
```

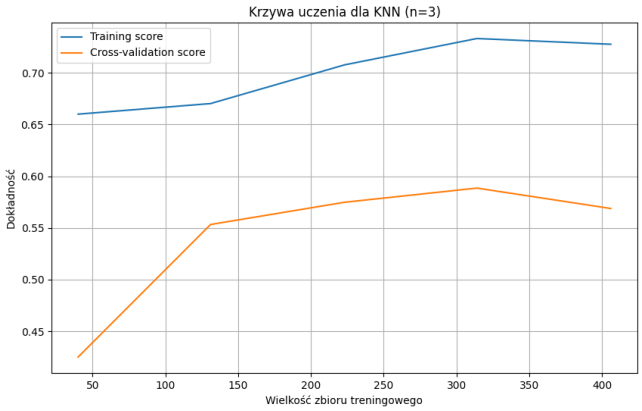
2.2 Klasyfikacja za pomocą KNN

Klasyfikator KNN (K-Nearest Neighbors) to jedna z najprostszych metod klasyfikacji w uczeniu maszynowym. Działa na zasadzie uczenia przez analogię, czyli przypisuje nowe dane do kategorii na podstawie podobieństwa do innych danych trenujących. Jak działa KNN: Kluczowym parametrem w KNN jest liczba sąsiadów (K), która określa, ile najbliższych sąsiadów będzie brane pod uwagę przy przypisywaniu etykiety do nowych danych. Dla nowego punktu danych, KNN oblicza odległość od tego punktu do każdego punktu danych w zbiorze treningowym. Następnie KNN wybiera K najbliższych sąsiadów nowego punktu danych na podstawie obliczonych odległości. Na podstawie etykiet tych K najbliższych sąsiadów, nowy punkt danych jest klasyfikowany do danej kategorii na podstawie najczęściej występującej etykiety wśród tych sąsiadów.

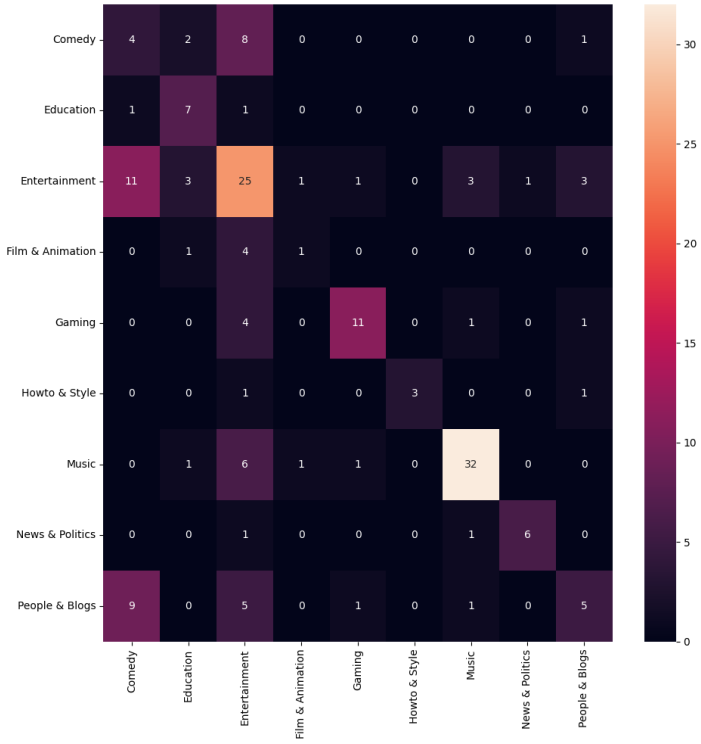
2.2.1 KNN dla K=3

```
# Klasyfikacja za pomocą KNN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Przewidywanie dla zestawu testowego
y_pred = knn.predict(X_test)

# Ewaluacja modelu
accuracy = accuracy_score(y_test, y_pred)
accuracy_knn3 = accuracy
print(f"Accuracy for n=3: {accuracy * 100:.2f}%")
# Output:
# Accuracy for n=3: 55.29%
```

Algorytm KNN dla K=3 uzyskał wynik na poziomie 55.29 %, na rysunkach 1 oraz 2 przedstawiono krzywą uczenia oraz macierz błędów dla tego algorytmu.



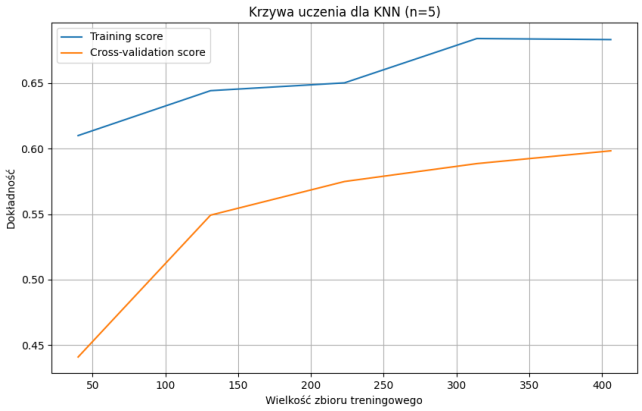
Rysunek 1: Krzywa uczenia dla knn3



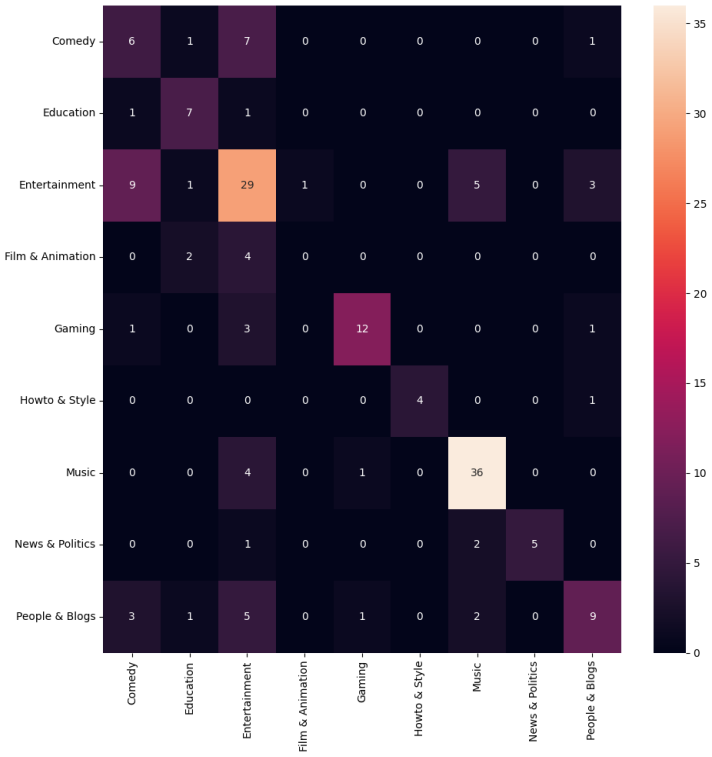
Rysunek 2: Macierz błędów dla knn3

2.2.2 KNN dla K=5

Algorytm KNN dla K=5 uzyskał wynik na poziomie 63.53 %, na rysunkach 3 oraz 4 przedstawiono krzywą uczenia oraz macierz błędów dla tego algorytmu.



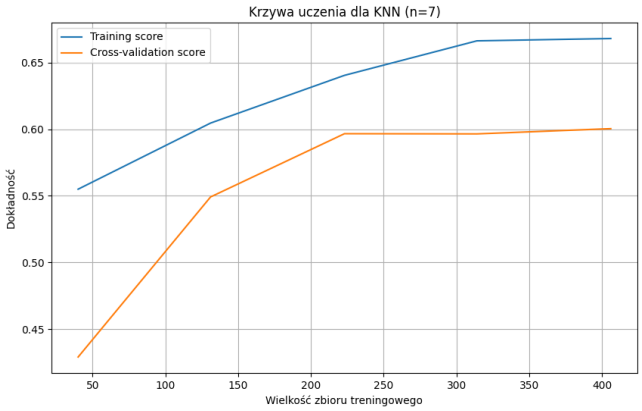
Rysunek 3: Krzywa uczenia dla knn5



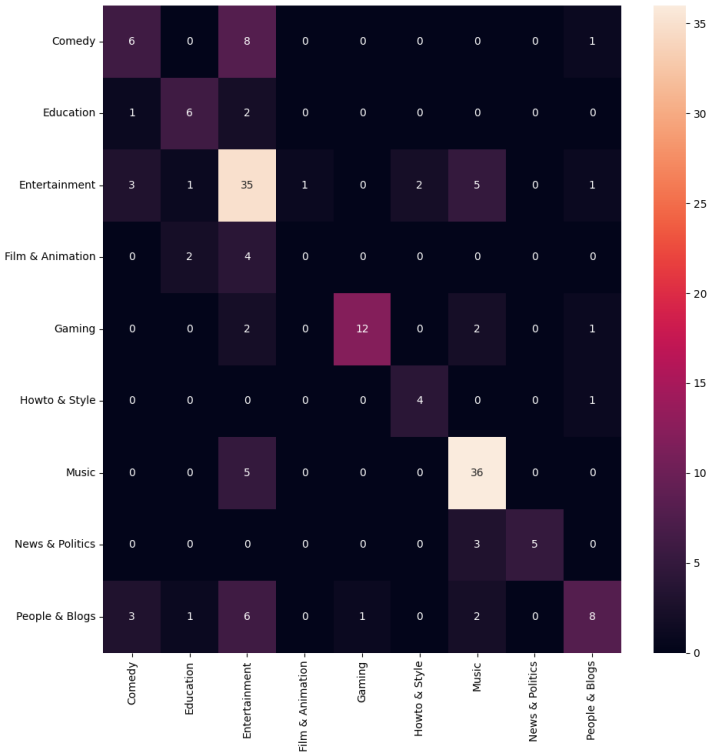
Rysunek 4: Macierz błędów dla knn5

2.2.3 KNN dla K=7

Algorytm KNN dla K=7 uzyskał wynik na poziomie 65.88 %, na rysunkach 5 oraz 6 przedstawiono krzywą uczenia oraz macierz błędów dla tego algorytmu.



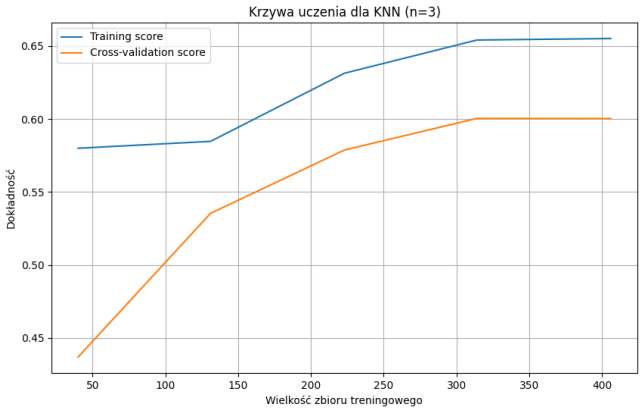
Rysunek 5: Krzywa uczenia dla knn7



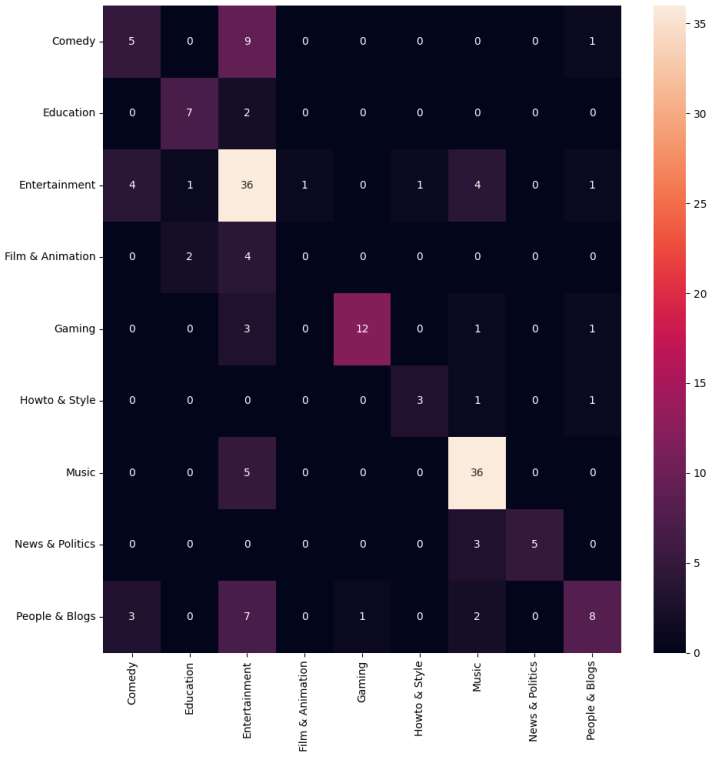
Rysunek 6: Macierz błędów dla knn7

2.2.4 KNN dla K=9

Algorytm KNN dla K=9 uzyskał wynik na poziomie 65.88 %, na rysunkach 7 oraz 8 przedstawiono krzywą uczenia oraz macierz błędów dla tego algorytmu.



Rysunek 7: Krzywa uczenia dla knn9



Rysunek 8: Macierz błędów dla knn9

2.3 Naive Bayes

Klasyfikator Naive Bayes jest probabilistycznym modelem uczenia maszynowego, który opiera się na twierdzeniu Bayesa. Jest to prosty, ale skuteczny algorytm do klasyfikacji, który jest często stosowany w analizie danych, klasyfikacji tekstu, filtracji spamu, itp.

Twierdzenie Bayesa: Naive Bayes opiera się na twierdzeniu Bayesa, które mówi, jakie jest prawdopodobieństwo wystąpienia danej klasy, gdy znane są pewne cechy. Matematycznie wyraża się to wzorem:

$$P(C|X) = \frac{P(X|C) \times P(C)}{P(X)}$$

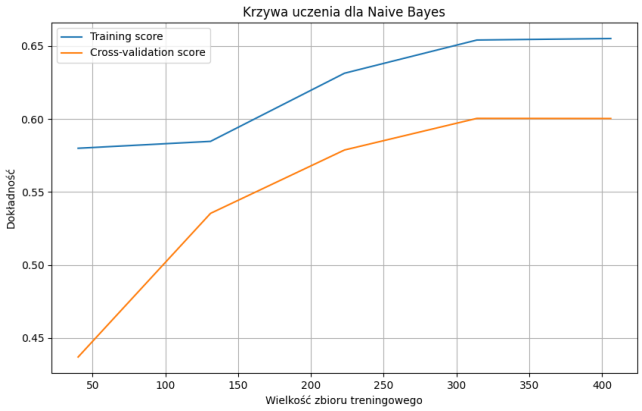
gdzie:

- $P(C|X)$ oznacza prawdopodobieństwo, że dana próbka należy do klasy C, pod warunkiem, że znane są cechy X.
- $P(X|C)$ to prawdopodobieństwo wystąpienia cech X, pod warunkiem, że próbka należy do klasy C.
- $P(C)$ to prawdopodobieństwo wystąpienia klasy C.
- $P(X)$ to prawdopodobieństwo wystąpienia cech X.

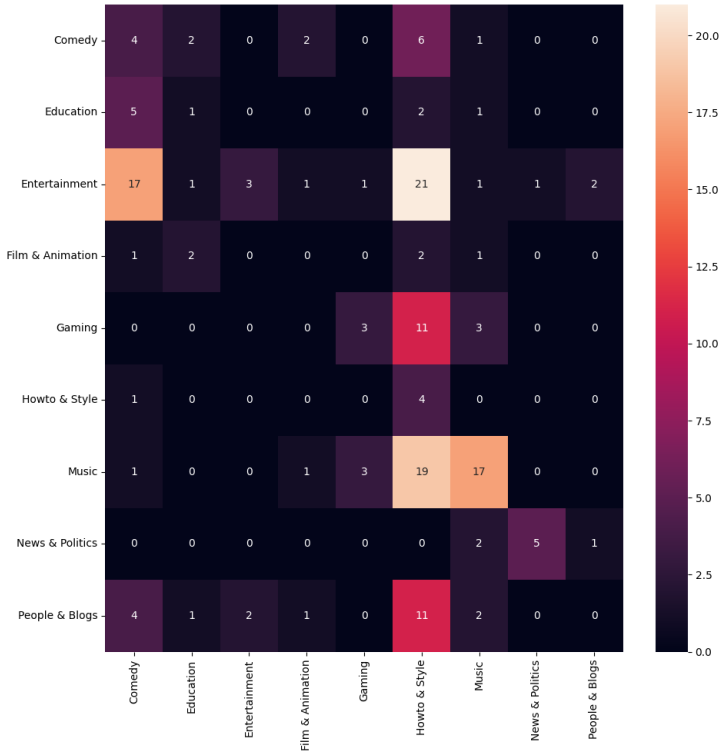
Naivety (naiwność): Klasyfikator Naive Bayes zakłada, że wszystkie cechy są niezależne od siebie. Oznacza to, że prawdopodobieństwo wystąpienia cech X pod warunkiem danej klasy C można rozłożyć jako iloczyn prawdopodobieństw wystąpienia pojedynczych cech.
Trening modelu: Podczas treningu modelu, Naive Bayes oblicza prawdopodobieństwa $P(X|C)$ dla każdej klasy C na podstawie danych treningowych.
Klasyfikacja: Gdy przychodzi czas na klasyfikację nowych danych, Naive Bayes oblicza prawdopodobieństwo $P(C|X)$ dla każdej klasy C przy użyciu twierdzenia Bayesa. Klasa z najwyższym prawdopodobieństwem jest przypisywana do nowych danych.

```python
# Naive Bayes
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)

accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naive Bayes Accuracy: {accuracy_nb * 100:.2f}%")
# Output:
# Naive Bayes Accuracy: 21.76%
```

Algorytm Naive Bayes uzyskał wynik na poziomie 21.76 %, na rysunkach 9 oraz 10 przedstawiono krzywą uczenia oraz macierz błędów dla tego algorytmu.



Rysunek 9: Krzywa uczenia dla Naive Bayes



Rysunek 10: Macierz błędów dla Naive Bayes

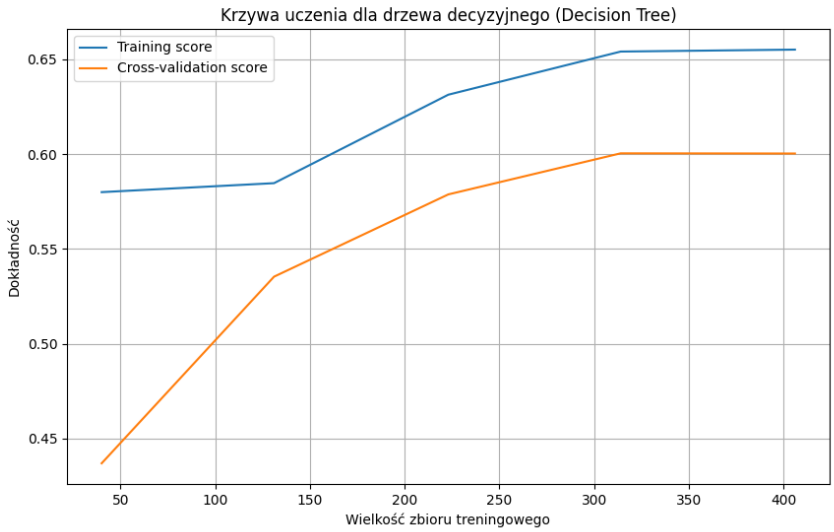
2.4 Drzewo decyzyjne (Decision Tree)

Drzewo decyzyjne jest jednym z najpopularniejszych modeli uczenia maszynowego, stosowanym zarówno do klasyfikacji, jak i regresji. Jest to model oparty na strukturze drzewa, gdzie każdy węzeł reprezentuje test na jednej z cech, każda krawędź jest wynikiem tego testu, a każdy liść reprezentuje etykietę klasy lub wartość prognozowaną.

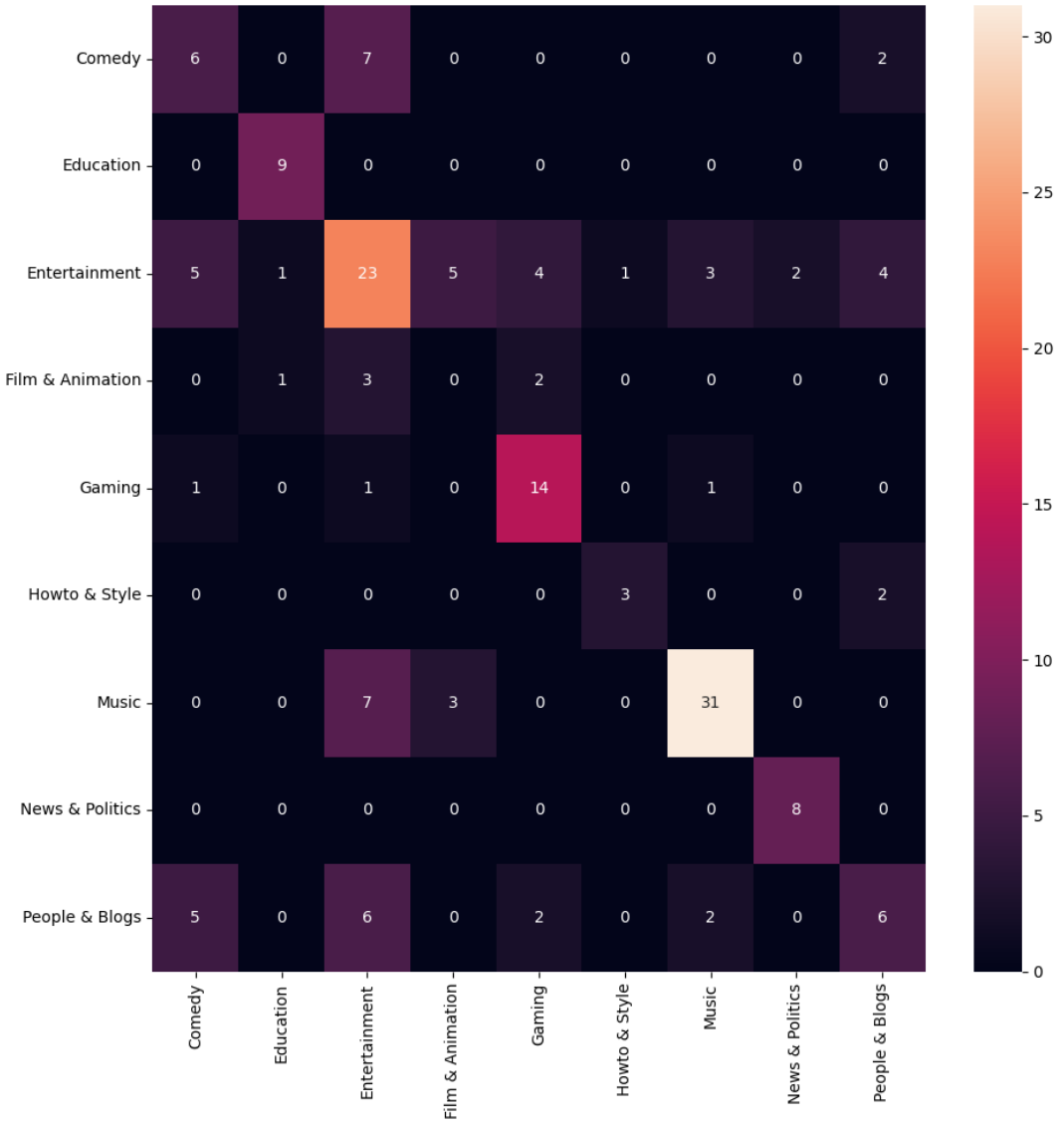
1. **Tworzenie drzewa**: Proces rozpoczyna się od korzenia drzewa, gdzie wybierana jest cecha, która najlepiej separuje dane. Proces ten jest powtarzany rekurencyjnie dla każdego podziału, aż osiągnięty zostanie warunek zatrzymania, np. maksymalna głębokość drzewa, minimalna liczba próbek w liściu, itp.
2. **Podział danych**: W każdym węźle drzewa dane są dzielone na podzbiory na podstawie wartości jednej z cech. Celem jest maksymalizacja jednorodności podgrup, np. minimalizacja entropii lub maksymalizacja czystości Gini.
3. **Budowa drzewa**: Proces budowy drzewa kontynuuje się aż do osiągnięcia warunku zatrzymania. Podczas tworzenia drzewa, algorytm wybiera cechy, które najlepiej separują dane, aby jak najlepiej dopasować model do danych treningowych.
4. **Klasyfikacja**: Gdy drzewo zostanie zbudowane, można użyć go do klasyfikacji nowych danych poprzez przeprowadzenie testów na cechach nowych danych i poruszając się wzdłuż gałęzi drzewa, aż do osiągnięcia liścia, który zawiera prognozowaną etykietę klasy.

```python
# drzewo decyzyjne
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)

accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Accuracy: {accuracy_dt * 100:.2f}%")
# Output:
# Decision Tree Accuracy: 58.82%
```

Algorytm uzyskał wynik na poziomie 58.82 %, na rysunkach 11 oraz 12 przedstawiono krzywą uczenia oraz macierz błędów dla tego algorytmu.



Rysunek 11: Krzywa uczenia dla Decision Tree



Rysunek 12: Macierz błędów dla Decision Tree

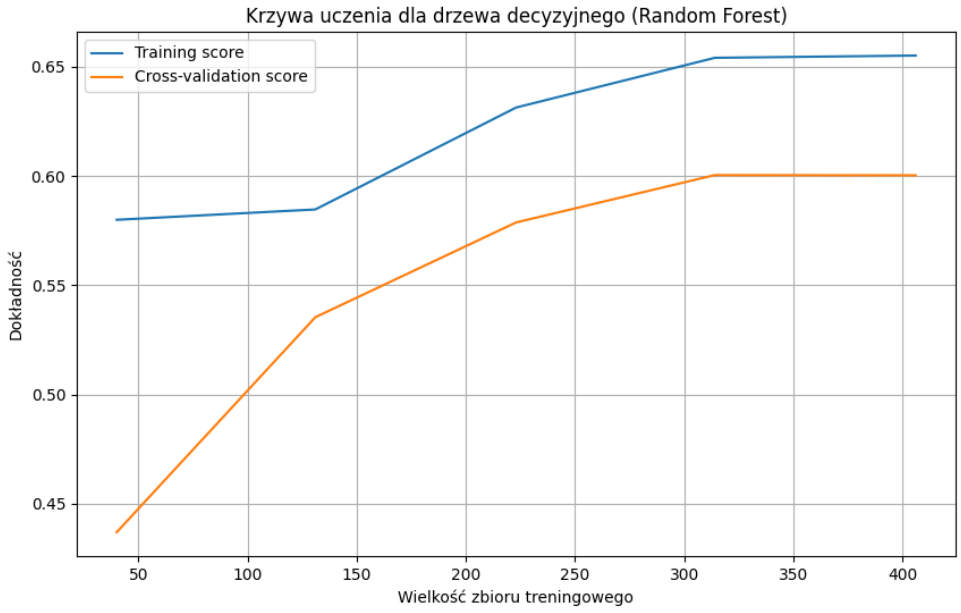
2.5 Drzewo decyzyjne (Random Forest)

Drzewo decyzyjne w Random Forest jest często stosowanym modelem uczenia maszynowego, który wykorzystuje wiele drzew decyzyjnych do klasyfikacji lub regresji. Jest to technika zespołowa (ensemble learning), która łączy wiele modeli drzew decyzyjnych, aby uzyskać lepszą wydajność predykcyjną niż pojedyncze drzewo decyzyjne.

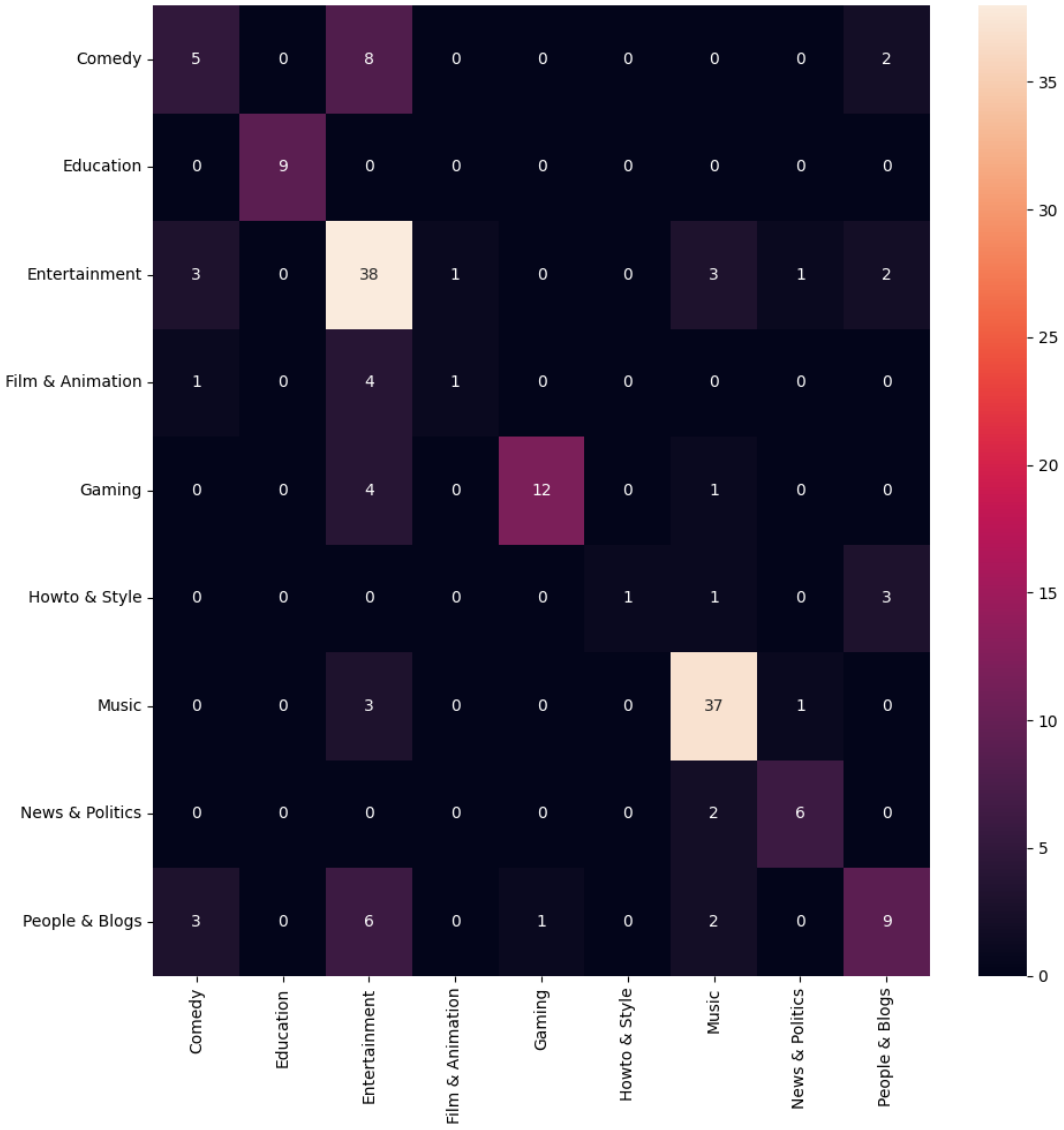
1. **Tworzenie drzew decyzyjnych**: Random Forest tworzy wiele drzew decyzyjnych na podstawie losowych podzbiorów danych treningowych i losowych podzbiorów cech. Każde drzewo jest trenowane niezależnie od innych drzew.
2. **Losowy wybór cech**: Podczas tworzenia każdego drzewa, Random Forest losowo wybiera podzbiór cech z pełnego zestawu cech. Ten proces nazywany jest baggingiem.
3. **Klasyfikacja lub regresja**: Gdy wszystkie drzewa zostały utworzone, Random Forest łączy wyniki z każdego drzewa w celu klasyfikacji lub regresji nowych danych. W przypadku klasyfikacji, często stosuje się głosowanie większościowe, a w przypadku regresji - średnią wartość prognozowaną.

```python
# Przykład użycia Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf * 100:.2f}%")
# Output:
# Random Forest Accuracy: 69.41%
```

Algorytm uzyskał wynik na poziomie 69.41 %, na rysunkach 13 oraz 14 przedstawiono krzywą uczenia oraz macierz błędów dla tego algorytmu.



Rysunek 13: Krzywa uczenia dla Random Forest



Rysunek 14: Macierz błędów dla Random Forest

2.6 Sieć neuronowa

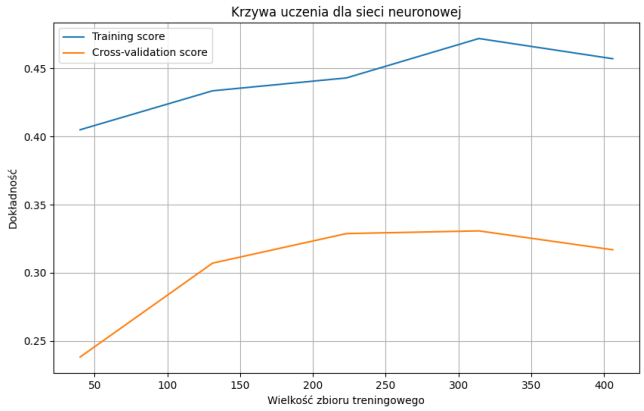
Przygotowanie danych Kategoria kanału jest kodowana za pomocą LabelEncodera,tak jak dla poprzednich algorytmów. Natomiast "channel_type"jest zakodowany za pomocą OneHotEncodera, który przekształca dane tekstowe w wektor binarny, ponieważ takiej formy oczekuje sieć neuronowa na wejściu. Następnie dane są podzielone na zestawy treningowe i testowe, a także skalowane za pomocą StandardScaler.

Definiowanie modelu sieci neuronowej Model sieci neuronowej jest zdefiniowany za pomocą biblioteki Keras. Składa się on z warstw gęsto połączonych neuronów. Pierwsza warstwa ma 64 neurony i funkcję aktywacji ReLU. Następnie następuje warstwa o 32 neuronach z funkcją aktywacji ReLU. Kolejna warstwa ma 16 neuronów z funkcją aktywacji softmax, która jest stosowana do klasyfikacji wieloklasowej. Ostatnia warstwa ma liczbę wyjść równą liczbie klas i funkcję aktywacji softmax.

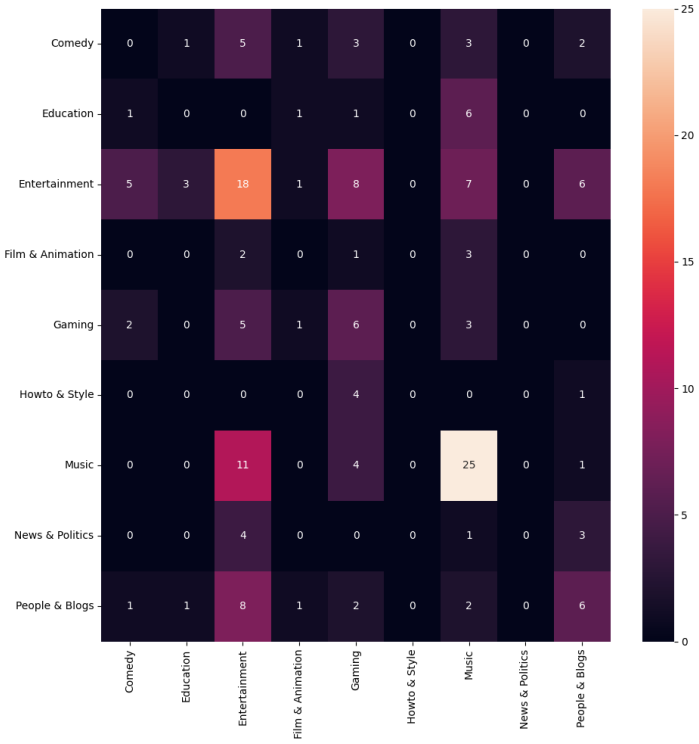
Kompilacja i trening modelu Model jest kompilowany za pomocą optymalizatora Adam i funkcji straty sparse categorical crossentropy, która jest używana, gdy etykiety są przekazywane w postaci całkowitoliczbowej. Następnie model jest trenowany na zestawie treningowym przez 1000 epok z rozmiarem wsadu 32 i walidacją na 20% danych treningowych.

```
le = LabelEncoder()
df['category'] = le.fit_transform(df['category'])
ohe = OneHotEncoder()
df['channel_type'] = ohe.fit_transform(df[['channel_type']]).toarray()

X = df[['uploads', 'video_views', 'video_views_for_the_last_30_days',
'subscribers','subscribers_for_last_30_days',
'channel_type']]
y = df['category']
X_scaled = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25,
random_state=42)


model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(16, activation="softmax"),
    Dense(len(le.classes_), activation='softmax')  # Liczba wyjść równa liczbie klas
])

# Kompilacja modelu
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Trening modelu
model.fit(X_train, y_train, epochs=1000, batch_size=32, validation_split=0.2)

# Przewidywanie dla zestawu testowego
y_pred = model.predict(X_test)

# Ewaluacja modelu
accuracy = accuracy_score(y_test, np.argmax(y_pred, axis=1))
print(f"Neural Network Accuracy: {accuracy * 100:.2f}%")
# Output:
# Neural Network Accuracy: 32.35%
```

Sieć uzyskała wynik na poziomie 32.35 %, na rysunkach 15 oraz 16 przedstawiono krzywą uczenia oraz macierz błędów dla tego algorytmu.



Rysunek 15: Krzywa uczenia dla Sieci neuronowej



Rysunek 16: Macierz błędów dla Sieci Neuronowej

3 Reguły asocjacyjne

Poniższy kod implementuje algorytm Apriori do analizy reguł asocjacyjnych.

Odczytanie danych Najpierw dane są wczytywane z pliku CSV za pomocą funkcji `pd.read_csv`. Następnie kolumny tekstowe są przekształcane na format binarny, gdzie wartość `True` oznacza, że dana kolumna zawiera dane, a `False` w przeciwnym przypadku.

Przygotowanie danych Kolumny tekstowe i numeryczne są przekształcane na zmienne binarne za pomocą funkcji `get_dummies`. W ten sposób tworzona jest macierz binarna, która będzie wykorzystywana do analizy algorytmem Apriori.

Zastosowanie algorytmu Apriori Algorytm Apriori jest stosowany do znalezienia częstych zbiorów elementów w danych za pomocą funkcji `apriori`. Parametr `min_support` określa minimalne wsparcie dla zestawów elementów.

Generowanie reguł asocjacyjnych Na podstawie częstych zbiorów elementów generowane są reguły asocjacyjne za pomocą funkcji `association_rules`. Parametr `metric` określa miarę oceny reguł, w tym przypadku używamy metryki `lift`.

Zapisanie reguł do pliku CSV Ostatecznie, wygenerowane reguły asocjacyjne są zapisywane do pliku CSV za pomocą funkcji `to_csv`.

```
text_columns = ['youtuber', 'category', 'title', 'country', 'abbreviation',
'channel_type']
for col in text_columns:
    df[col] = df[col].astype(str).apply(lambda x: 1 if x else 0).astype(bool)


numeric_columns = ['rank', 'subscribers', 'video_views', 'video_views_rank',
'country_rank','video_views_for_the_last_30_days',
                   'lowest_monthly_earnings', 'highest_monthly_earnings',
                'lowest_yearly_earnings','highest_yearly_earnings',
                   'subscribers_for_last_30_days','population'
                   'gross_tertiary_education_enrollment',
        'unemployment_rate','urban_population']

for col in numeric_columns:
    df[col] = df[col].astype(str).apply(lambda x: 1 if x else 0).astype(bool)
# Stworzenie DataFrame z dummy variables dla kolumn tekstowych oraz numerycznych
basket = pd.get_dummies(df[text_columns + numeric_columns], drop_first=True)

# Zastosowanie algorytmu Apriori
frequent_itemsets = apriori(basket, min_support=0.11, use_colnames=True)

# Generowanie reguł asocjacyjnych
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.head()

# Zapisanie reguł do pliku CSV
rules.to_csv('association_rules.csv', index=False)
```

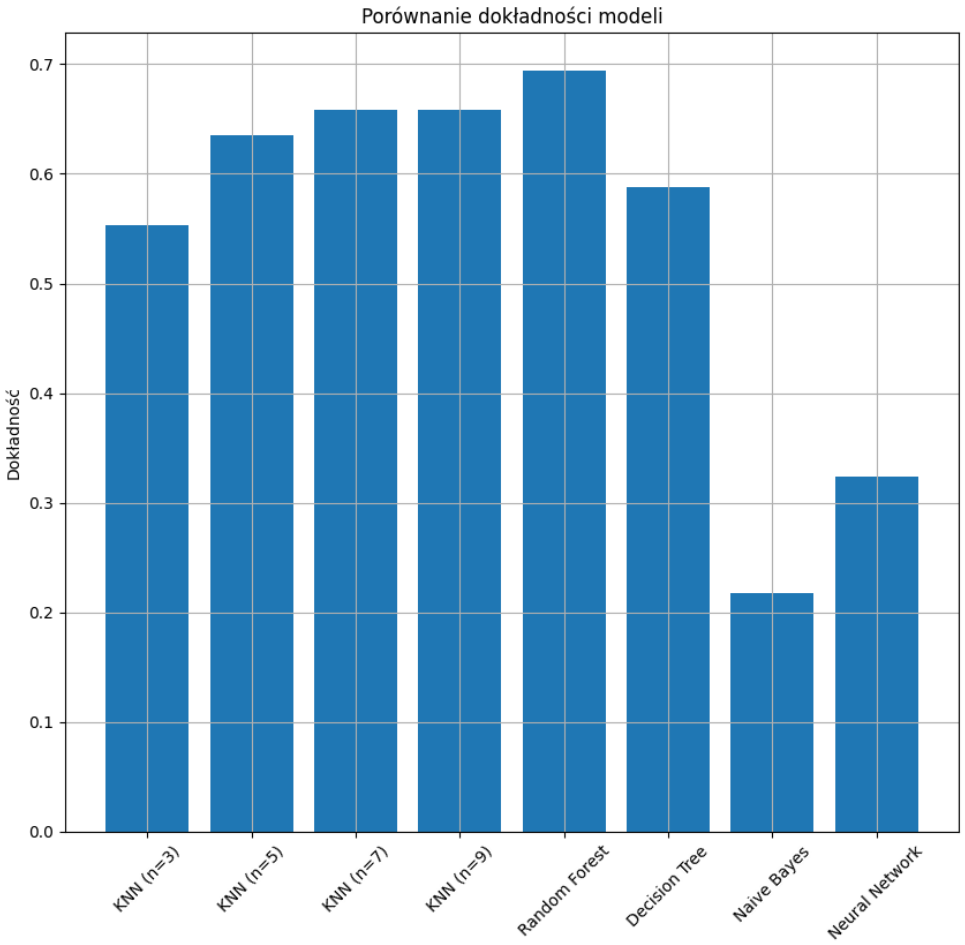
Wyniki sugerują, że wszystkie reguły asocjacyjne mają maksymalne wartości dla wielu miar, takich jak wsparcie, pewność, podwajanie i mierzenie Zhang'a. Jednakże, taki wynik może być rezultatem zbyt małej ilości danych wejściowych lub niewystarczającej zróżnicowania w danych.

Brak różnorodności danych Jednym z możliwych powodów takich wyników może być brak różnorodności w danych wejściowych. Jeśli wszystkie reguły mają maksymalne wartości dla wszystkich miar, oznacza to, że dane wejściowe prawdopodobnie nie zawierają wystarczającej ilości lub różnorodności informacji, aby umożliwić algorytmowi Apriori znalezienie interesujących reguł asocjacyjnych.

Zbyt mała ilość danych Innym potencjalnym powodem jest zbyt mała ilość danych wejściowych. W przypadku, gdy ilość danych jest ograniczona, algorytm Apriori może mieć ograniczone możliwości wykrywania istotnych wzorców w danych. Może to prowadzić do sytuacji, w której wszystkie reguły mają maksymalne wartości dla wszystkich miar, pomimo braku rzeczywistych wzorców w danych.

4 Podsumowanie

W ramach niniejszego raportu przeprowadzono analizę porównawczą różnych algorytmów uczenia maszynowego w kontekście klasyfikacji danych. Badanie miało na celu zidentyfikowanie najbardziej skutecznej metody klasyfikacji w stosunku do zadanego zestawu danych oraz zrozumienie charakterystyki i działania poszczególnych algorytmów. W ramach eksperymentu porównano osiem różnych algorytmów klasyfikacji, w tym KNN z różnymi wartościami parametru n (3, 5, 7, 9), Random Forest, Decision Tree, Naive Bayes oraz Sieć Neuronową. Wyniki eksperymentu, widoczne na rysunku 17 pokazały, że Random Forest osiągnął najlepsze wyniki w porównaniu do pozostałych algorytmów. Random Forest wykazał się najwyższą skutecznością w klasyfikacji danych, osiągając najwyższe accuracy. W przeciwieństwie do Random Forest, Naive Bayes wypadł najgorzej pod względem skuteczności klasyfikacji. Wyniki tego algorytmu były znacząco niższe niż pozostałych metod. W trakcie przygotowania raportu badawczego przeprowadzono szereg analiz i eksperymentów, co pozwoliło lepiej zrozumieć i porównać różne techniki uczenia maszynowego. Pozwoliło to także na zgłębienie wiedzy na temat przetwarzania danych, dostosowywania parametrów algorytmów oraz interpretacji wyników eksperymentów.



Rysunek 17: Porównanie dokładności modeli

Wykorzystane źródła

Algorytm KNN

Random Forest

Decision Tree

Naive Bayes

Reguły asocjacyjne