

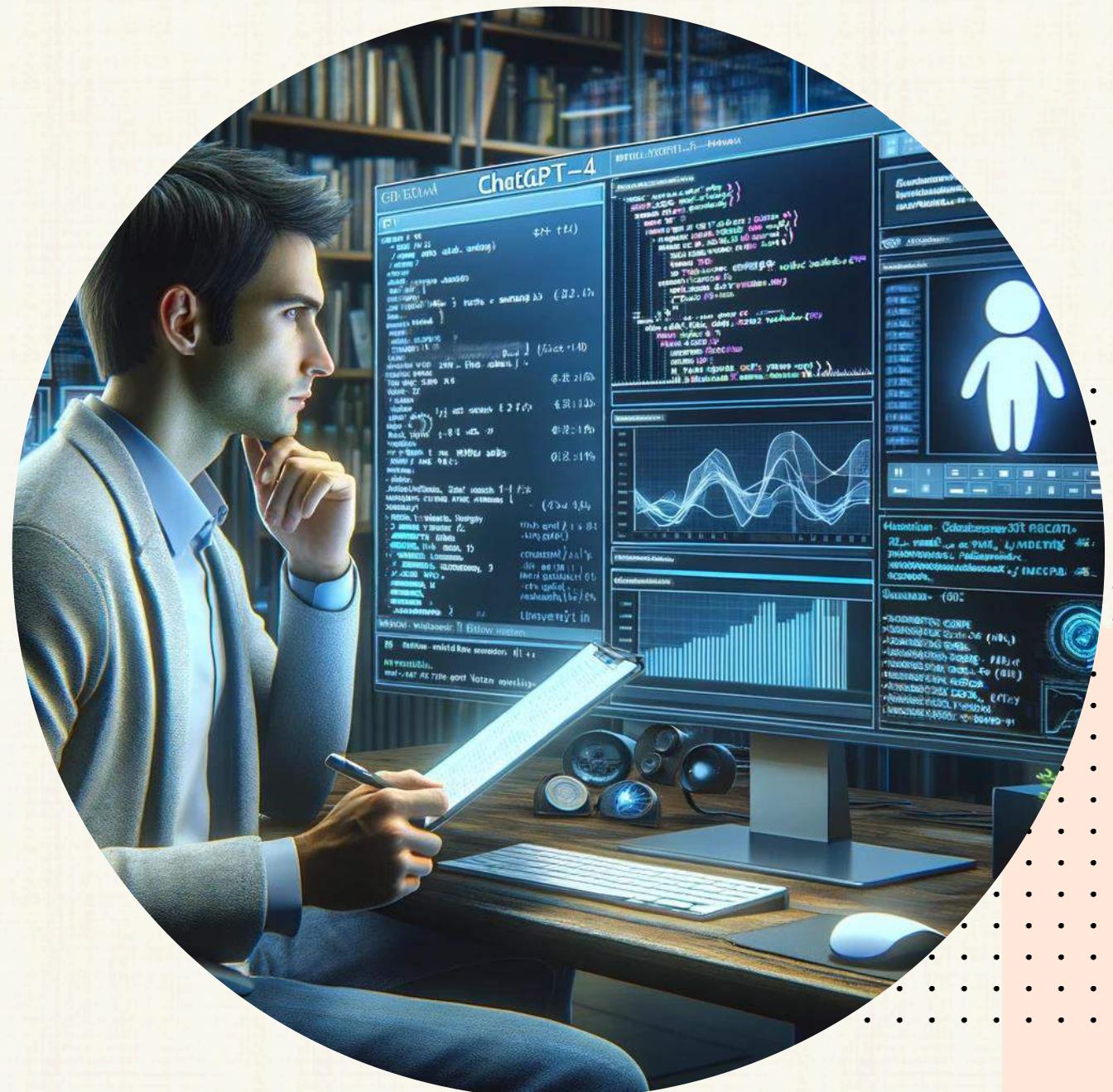
# PROGRAMOWANIE Z UŻYCIEM WIELKICH MODELI JĘZYKOWCH



# CEL PRACY:

Ocena jak model językowy ChatGPT-4 Radzi sobie z problemami uczenia maszynowego.

Czyli oceana pod względem wsparcia w generowaniu kodu, wyboru rozwiązań, interpretacji wyników i końcowego modelu który utworzyła.



# CO TO CHATGPT- 4

Chat

4

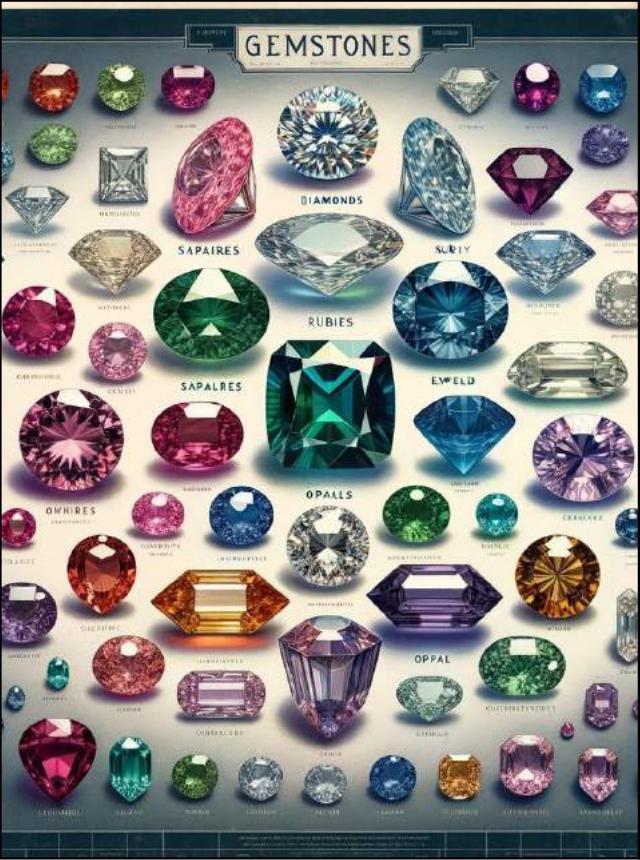
ChatGPT-4 to zaawansowany model językowy opracowany przez OpenAI, który bazuje na architekturze GPT-4 (Generative Pre-trained Transformer 4). Jest to system uczenia maszynowego, który został wytrenowany na ogromnym zbiorze danych tekstowych i jest zdolny do generowania koherentnego i kontekstualnego tekstu, odpowiadania na pytania, a także wykonywania zadań takich jak tłumaczenie języków czy pisanie kodu programistycznego. ChatGPT-4 wyróżnia się jeszcze większą zdolnością do zrozumienia złożonych zapytań i dostarczania bardziej precyzyjnych odpowiedzi niż jego poprzednie wersje, dzięki czemu znajduje zastosowanie w wielu dziedzinach, od edukacji po rozwój oprogramowania.

# CO TO UCZENIE MASZYNOWE

Uczenie maszynowe to dziedzina sztucznej inteligencji, która polega na tworzeniu algorytmów zdolnych do uczenia się i dokonywania prognoz lub decyzji na podstawie danych. Wykorzystuje statystyczne metody do dawania komputerom zdolności "uczenia się" z danych, bez bezpośredniego programowania specyficznych zadań. Dzięki temu, systemy uczenia maszynowego mogą automatycznie poprawiać swoją wydajność w miarę gromadzenia i analizowania coraz większej ilości danych.

# WYKONANE PROJEKTY:

A. KLASYFIKACJA  
KAMIENI  
SZLACZETNYCH



B. REGRESJA CEN  
UŻYWANYCH  
SAMOCHODÓW



C. SYSTEM  
REKOMENDACJI  
FILMÓW NETFLIX



D. TRENOWANIE  
AGENTA  
KUNGFUMASTER



# KLASYFIKACJA KAMIENI SZLACHTNYCH



# PROMPT STARTOWY



Ja:

Cześć, chciałbym, abyś wytrenował model uczenia maszynowego w Pythonie do przewidywania kategorii kamieni szlachetnych. Mój zbiór danych zawiera ponad 3 200 obrazów kamieni szlachetnych. Obrazy są podzielone na 87 klas, które zostały już przetworzone na dane treningowe i testowe. Obrazy mają różne rozmiary i są w formacie .jpeg.

Ten zbiór danych składa się z dwóch folderów:

**train:** Ten folder zawiera 87 podfolderów i łącznie około 2 800 plików. Każdy podfolder zawiera obrazy .jpeg różnych kamieni szlachetnych.

**test:** Ten folder zawiera 87 podfolderów i łącznie około 400 plików. Każdy podfolder zawiera obrazy .jpeg różnych kamieni szlachetnych.

Ścieżka pliku do moich danych treningowych: "C:\Users\mikoł\aDesktop\Pulpit\Praca Inżynierska\Uczenie Maszynowe\Klasyfikacja\data\train"

Ścieżka pliku do moich danych testowych: "C:\Users\mikoł\aDesktop\Pulpit\Praca Inżynierska\Uczenie Maszynowe\Klasyfikacja\data\test"



ChatGPT:

Dzięki za informacje. Aby zbudować model klasyfikacji obrazów kamieni szlachetnych, wykorzystamy bibliotekę TensorFlow wraz z Keras. Poniżej znajduje się krok po kroku, jak to zrobić:

1. **Wczytywanie i Przetwarzanie Danych**

Będziemy korzystać z `ImageDataGenerator` do wczytywania obrazów, a także do zastosowania augmentacji danych<sup>[14]</sup> (wzmacnianie danych poprzez modyfikację oryginalnych obrazów).

2. **Budowanie Modelu**

Do klasyfikacji obrazów użyjemy sieci konwolucyjnej<sup>[15]</sup> (Convolutional Neural Network, CNN).

3. **Trening Modelu**

Po zbudowaniu modelu przeprowadzimy proces treningu.

4. **Ewaluacja Modelu**

Ocenimy model na danych testowych.

Oto kod:

```
python Copy code

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Stałe
IMAGE_SIZE = 150
BATCH_SIZE = 32
EPOCHS = 10

# 1. Wczytywanie i Przetwarzanie Danych
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255) # Tylko skalowanie dla danych test

train_generator = train_datagen.flow_from_directory(
    "C:\Users\mikol\Desktop\Pulpit\Praca Inżynierska\Uczenie Maszynowe\Klasyfikacja",
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    "C:\Users\mikol\Desktop\Pulpit\Praca Inżynierska\Uczenie Maszynowe\Klasyfikacja",
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical')
```

```
# 2. Budowanie Modelu
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(87, activation='softmax')])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# 3. Trening Modelu
history = model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=test_generator)

# 4. Ewaluacja Modelu
loss, accuracy = model.evaluate(test_generator)
print(f"Strata testu: {loss:.4f}")
print(f"Dokładność testu: {accuracy:.4f}")
```

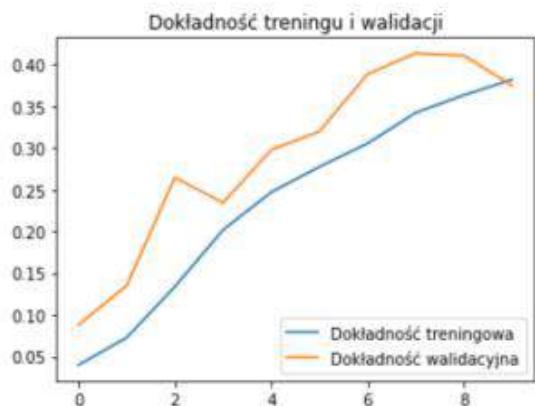
Kilka uwag:

- Aby dostosować ten kod do Twojego przypadku, możesz zmienić wartości, takie jak `IMAGE\_SIZE`, `BATCH\_SIZE` czy `EPOCHS` w zależności od Twoich potrzeb.
- Jeśli chodzi o architekturę sieci, możesz ją dostosować w zależności od wyników, jakie uzyskasz.
- Możesz również dodać checkpointy, zapisywanie modelu itp., aby ulepszyć proces treningu.

# OCENA TRENINGU

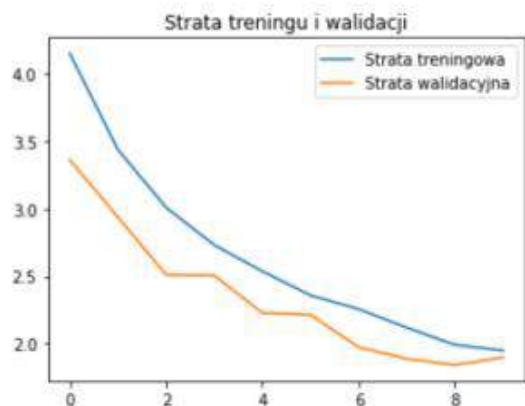


Ja:  
Czy te wyniki są satysfakcjonujące?



Strata testu: 1.8988

Dokładność testu: 0.3747



ChatGPT:  
Analizując wykresy oraz podane statystyki:

- Dokładność (Accuracy):** Model osiągnął dokładność na poziomie około 38% na zbiorze walidacyjnym po 10 epokach. To jest stosunkowo niska wartość, zwłaszcza biorąc pod uwagę, że klasyfikujesz obrazy do 87 różnych klas.
- Strata (Loss):** Strata na zbiorze treningowym maleje w miarę postępu treningu, co jest pożądaną tendencją. Jednakże strata na zbiorze walidacyjnym, choć generalnie maleje, zaczyna się nieznacznie zwiększać w ostatnich epokach, co może wskazywać na początek przetrenowania.
- Wykresy:** Z wykresów można zauważyc, że dokładność walidacyjna jest generalnie niższa niż dokładność treningowa, co jest typowe, ale różnica nie jest drastyczna. Wskaźniki straty wydają się być bardziej stabilne.

Podsumowując, wyniki nie są jeszcze satysfakcjonujące. Istnieje kilka kroków, które można podjąć w celu poprawy wyników:

- Zwiększenie liczby epok:** Model może potrzebować więcej czasu na naukę, zwłaszcza przy takiej liczbie klas. Możesz spróbować trenować przez więcej epok, obserwując, czy dokładność walidacyjna się poprawia.
- Zmiana architektury modelu:** Możesz eksperymentować z liczbą i rozmiarem warstw w modelu.
- Data Augmentation:** Zastosowanie technik augmentation danych może pomóc modelowi w generalizacji i zapobieganiu przetrenowaniu.
- Regularizacja:** Oprócz dropoutu, możesz rozważyć dodanie regularizacji L1 lub L2 do modelu.
- Zastosowanie wczesnego zatrzymywania (Early Stopping):** Możesz zastosować wczesne zatrzymywanie, aby automatycznie przerwać trening, gdy dokładność walidacyjna przestaje się poprawiać.
- Transfer Learning<sup>[13]</sup>:** Wykorzystaj wytrenowane modele, takie jak VGG16 lub ResNet, do ekstrakcji cech, a następnie dodaj kilka warstw własnych, które zostaną wytrenowane na Twoim zbiorze danych.

# POPRAWA MODELU



Ja:

Dokonaj zmian w podanym przez ciebie kodzie które twoim zdaniem najlepiej wpłynęłyby na wyniki.



ChatGPT:

Okej, skupmy się na kilku kluczowych aspektach, które mogą poprawić wyniki:

1. **Transfer Learning:** Wykorzystamy model VGG16 jako bazę, co pozwoli nam na skorzystanie z wstępnie wytrenowanych wag dla ekstrakcji cech. Następnie dodamy kilka własnych warstw do klasifikacji.
2. **Data Augmentation:** Aby zwiększyć różnorodność danych treningowych i zapobiec przetrenowaniu.
3. **Early Stopping:** Automatyczne zatrzymywanie treningu, gdy wyniki walidacji zaczynają się pogarszać.

```
# Wczesne zatrzymywanie
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=[early_stop])
```

Oto zmodyfikowany kod:

python

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping

# Wczytanie modelu VGG16 bez górnych warstw
base_model = VGG16(include_top=False, weights='imagenet', input_shape=(150, 150, 3))

# Zamrożenie wag w modelu bazowym
for layer in base_model.layers:
    layer.trainable = False

# Dodanie własnych warstw
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
x = Dense(87, activation='softmax')(x)

model = Model(base_model.input, x)

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

# TECHNIKI KLASYFIKACJI



Ja:

Kod działa już poprawnie. Jakie znasz techniki do testowania wydajności modelu? Podaj mi kod dla jednej z nich.



ChatGPT:

Testowanie wydajności modelu klasyfikacji obrazów można przeprowadzić na wiele sposobów, zależnie od tego, które aspekty wydajności są najbardziej interesujące. Oto kilka technik:

1. **Confusion Matrix<sup>[14]</sup>:** Macierz błędów pozwala zrozumieć, które klasy są mylone przez model.
2. **Classification Report:** Zawiera podsumowanie miar wydajności, takich jak precyzyja, pełność (recall), f1-score dla każdej klasy.
3. **ROC Curves i AUC:** Są używane głównie w klasyfikacji binarnej do oceny wydajności modelu.
4. **Precision-Recall Curves:** Podobnie jak ROC, ale często używane w przypadkach, gdy klasy są niezbalansowane.

python

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap = 'Blues_r'
plt.ylabel('Rzeczywista etykieta')
plt.xlabel('Przewidywana etykieta')
title = 'Dokładność: {}'.format(accuracy_score(true_classes, predicted_classes))
plt.title(title, size = 12)
plt.show()
```

Oto przykład jak użyć Classification Report i Confusion Matrix:

python

Copy code

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Przewidzenia dla danych testowych
test_generator.reset() # Resetowanie generatora przed przewidywaniami
predictions = model.predict(test_generator, steps=test_generator.n // test_generator.batch_size)
predicted_classes = np.argmax(predictions, axis=1)

# Prawdziwe etykiety
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Classification Report
report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print(report)

# Confusion Matrix
cm = confusion_matrix(true_classes, predicted_classes)
print(cm)
```



# REGRESJA CEN UŻYWANYCH SAMOCHODÓW Z WIELKIEJ BRYTANII

REGRESJA CEN UŻYWANYCH  
SAMOCHODÓW

# P R O M P T   S T A R T O W Y



Ja:

Cześć, chciałbym, abyś wytrenował model uczenia maszynowego do przewidywania zmiennej 'Price'. Moje dane są w załączniku.



## used\_cars\_UK.csv Spreadsheet



 ChatGPT:

## python

 [Copy code](#)

```
import pandas as pd

# Load the dataset to check its contents
file_path = '/mnt/data/used_cars_UK.csv'
used_cars_df = pd.read_csv(file_path)

# Displaying the first few rows of the dataset to understand its structure
used_cars_df.head()
```

*Twoja baza danych używanych samochodów z Wielkiej Brytanii zawiera następujące kolumny:*

1. **'Unnamed: 0'**: Wygląda na indeks wierszy.
  2. **'title'**: Nazwa/model samochodu.
  4. **'Price'**: Cena samochodu (zmienna, którą chcesz przewidzieć).
  5. **'Mileage(miles)'**: Przebieg w milach.
  6. **'Registration\_Year'**: Rok rejestracji.
  7. **'Previous Owners'**: Liczba poprzednich właścicieli.
  8. **'Fuel type'**: Typ paliwa.
  9. **'Body type'**: Typ nadwozia.
  10. **'Engine'**: Pojemność silnika.
  11. **'Gearbox'**: Typ skrzyni biegów.
  12. **'Doors'**: Liczba drzwi.
  13. **'Seats'**: Liczba siedzeń.
  14. **'Emission Class'**: Klasa emisji.
  15. **'Service history'**: Historia serwisowa.

Aby wytrenować model uczenia maszynowego do przewidywania ceny (`Price`), należy wykonać następujące kroki:

1. **Czyszczenie Danych:** Usunięcie lub uzupełnienie brakujących danych, przekształcenie tekstowych zmiennych kategorialnych na numeryczne.
2. **Eksploracja Danych:** Analiza rozkładów zmiennych, sprawdzenie korelacji między zmiennymi.
3. **Inżynieria Cech<sup>[15]</sup> (Feature Engineering):** Utworzenie nowych cech, które mogą lepiej reprezentować dane dla modelu.
4. **Podział Danych:** Podzielenie danych na zestawy treningowe i testowe.
5. **Wybór Modelu:** Wybór odpowiedniego modelu uczenia maszynowego.
6. **Trenowanie Modelu:** Uczenie modelu na danych treningowych.
7. **Ocena Modelu:** Testowanie modelu na danych testowych i ocena jego dokładności.
8. **Dostrojenie Modelu:** Dostosowanie parametrów modelu dla lepszej wydajności.

Przystąpię teraz do przetworzenia danych i przygotowania ich do modelowania. Zacznę od czyszczenia danych.

Analiza brakujących danych i typów danych w Twoim zbiorze danych wykazała:

1. **Brakujące dane:**
  - `Previous Owners`: Brakuje 1409 wartości.
  - `Engine`: Brakuje 45 wartości.
  - `Doors`: Brakuje 25 wartości.
  - `Seats`: Brakuje 35 wartości.
  - `Emission Class`: Brakuje 87 wartości.
  - `Service history`: Brakuje 3145 wartości.
2. **Typy danych:**
  - Większość kolumn ma odpowiednie typy danych, takie jak liczby całkowite (`int64`) dla ceny, przebiegu, roku rejestracji itd.
  - Kolumny z danymi tekstowymi, takie jak `title`, `Fuel type`, `Body type`, `Engine`, `Gearbox`, `Emission Class` i `Service history`, są typu `object` i mogą wymagać przekształcenia do formatu numerycznego dla modelowania.

python

```
# Checking for missing values and data types in the dataset
missing_values = used_cars_df.isnull().sum()
data_types = used_cars_df.dtypes

missing_values, data_types
```

 Copy code

Należy rozważyć następujące kroki w celu przygotowania danych:

1. **Usunięcie lub uzupełnienie brakujących wartości:** Możemy usunąć wiersze lub kolumny z dużą liczbą brakujących danych lub uzupełnić brakujące wartości, na przykład używając średniej, mediany lub najczęściej występującej wartości.
2. **Kodowanie zmiennych kategorialnych:** Zmienne tekstowe (kategorialne) należy przekształcić na numeryczne przy użyciu technik takich jak kodowanie gorąco-jedynkowe (one-hot encoding) lub kodowanie etykiet (label encoding).
3. **Normalizacja danych:** Normalizacja cech liczbowych, takich jak przebieg czy rok rejestracji, może być pomocna w niektórych modelach.

Zacznę od czyszczenia danych, skupiając się na uzupełnianiu brakujących wartości i przekształcaniu zmiennych kategorialnych.

```
python

from sklearn.impute import SimpleImputer
import numpy as np

# Dropping the 'Unnamed: 0' column as it's just an index
used_cars_df.drop(columns='Unnamed: 0', inplace=True)

# Imputing missing values
# For numerical columns, using median to impute missing values
num_imputer = SimpleImputer(strategy='median')
used_cars_df['Previous Owners'] = num_imputer.fit_transform(used_cars_df[['Previous Owners']])
used_cars_df['Doors'] = num_imputer.fit_transform(used_cars_df[['Doors']])
used_cars_df['Seats'] = num_imputer.fit_transform(used_cars_df[['Seats']])

# For categorical columns, using the mode (most frequent) to impute missing values
cat_imputer = SimpleImputer(strategy='most_frequent')
used_cars_df['Engine'] = cat_imputer.fit_transform(used_cars_df[['Engine']])
used_cars_df['Emission Class'] = cat_imputer.fit_transform(used_cars_df[['Emission Class']])
used_cars_df['Service history'] = cat_imputer.fit_transform(used_cars_df[['Service history']])

# Checking the dataset after imputation
used_cars_df.isnull().sum()
```

Brakujące dane w zbiorze zostały pomyślnie uzupełnione. Teraz przystąpię do kodowania zmiennych kategorialnych, aby umożliwić ich użycie w modelu uczenia maszynowego.

Użyję techniki kodowania gorąco-jedynkowego (one-hot encoding) dla zmiennych kategorialnych. Ta technika tworzy nowe kolumny wskazujące obecność (lub brak) każdej możliwej wartości w oryginalnej kolumnie danych. Jest to przydatne dla modeli, które wymagają danych numerycznych, takich jak większość algorytmów uczenia maszynowego.

```
python

from sklearn.preprocessing import OneHotEncoder

# Selecting categorical columns for one-hot encoding
categorical_columns = ['Fuel type', 'Body type', 'Engine', 'Gearbox', 'Emission Class']

# Applying OneHotEncoder
encoder = OneHotEncoder(sparse=False, drop='first') # 'drop=first' to avoid multicollinearity
encoded_columns = encoder.fit_transform(used_cars_df[categorical_columns])
encoded_columns_df = pd.DataFrame(encoded_columns, columns=encoder.get_feature_names_out())

# Adding the new encoded columns to the original dataframe
used_cars_encoded_df = used_cars_df.join(encoded_columns_df)

# Dropping the original categorical columns as they are now encoded
used_cars_encoded_df.drop(columns=categorical_columns, inplace=True)

# Displaying the modified dataframe
used_cars_encoded_df.head()
```

Zmienna kategorialna została pomyślnie zakodowana za pomocą kodowania gorąco-jedynkowego. Teraz mamy dodatkowe kolumny, które reprezentują różne kategorie dla każdej z oryginalnych zmiennych kategorialnych.

Następnym krokiem jest podział danych na zestawy treningowe i testowe. Zwykle stosuje się stosunek podziału, na przykład 80% danych do treningu i 20% do testowania. Następnie będziemy mogli wybrać i wytrenować model uczenia maszynowego, wykorzystując te dane.

Przystępuję teraz do podziału danych.

python

Copy code

```
from sklearn.model_selection import train_test_split

# Defining the features and the target variable
X = used_cars_encoded_df.drop(columns=['title', 'Price']) # Dropping the title column
y = used_cars_encoded_df['Price']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Checking the shape of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Dane zostały podzielone na zestawy treningowe i testowe. Zestaw treningowy składa się z 2948 próbek, a zestaw testowy z 737 próbek.

Teraz wybiorę model uczenia maszynowego do przewidywania ceny samochodu ('Price'). Biorąc pod uwagę charakter problemu (regresja) oraz rodzaj i rozmiar danych, dobrze nadają się następujące modele:

1. **Regresja Liniowa:** Prosty i efektywny dla problemów regresji liniowej.
2. **Drzewo Decyzyjne:** Dobra radzi sobie z nieliniowością i interakcjami między cechami.
3. **Las Losowy (Random Forest):** Bardziej zaawansowana wersja drzew decyzyjnych, zazwyczaj oferująca lepszą wydajność.
4. **Gradient Boosting:** Potężny model, który iteracyjnie poprawia swoje prognozy.

Rozpoczniemy od wytrenowania modelu regresji liniowej, a następnie ocenimy jego wydajność. W zależności od wyników, możemy rozważyć użycie bardziej zaawansowanych modeli.

python

Copy code

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Training the Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Predicting on the test set
y_pred = lin_reg.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mse, r2
```

Model regresji liniowej został wytrenowany i oceniony na zestawie testowym. Wyniki są następujące:

- Średni Błąd Kwadratowy (MSE): 4,891,755.08
- Współczynnik determinacji ( $R^2$ ): 0.775

Współczynnik  $R^2$  wynosi około 0.775, co oznacza, że model wyjaśnia około 77.5% zmienności ceny samochodu. Jest to dość dobra wydajność, ale możliwe, że inne modele, takie jak las losowy czy gradient boosting, mogłyby dać lepsze wyniki.

Jeśli chcesz, mogę przetestować inne modele, aby sprawdzić, czy mogą one zapewnić lepsze przewidywanie. Czy chciałbyś, żebym kontynuował z innymi modelami, czy też skupić się na dalszym dostosowywaniu i analizie modelu regresji liniowej?

# BŁĄD TRENOVANIA LEPSZYCH MODELI



Ja:

Dziękuję za wytrenowanie modelu regresji liniowej na tych danych. Co do twojego pytania dotyczącego innych modeli to tak. Jeśli masz możliwość to przetestuj też bardziej zaawansowane modele które mogą dać lepsze wyniki.



ChatGPT:

Wygląda na to, że wystąpił błąd techniczny podczas próby trenowania bardziej zaawansowanych modeli. Możliwe przyczyny to ograniczenia sprzętowe lub zasoby systemowe, które nie są wystarczające do obsługi tych modeli. Zamiast próbować ponownie, mogę podać ogólne informacje na temat tych modeli i jak zazwyczaj są używane do poprawy wyników w podobnych zadaniach:

1. **Random Forest (Las Losowy):** Jest to model ensemble, który działa przez trenowanie wielu drzew decyzyjnych na różnych podzbiorach danych i uśrednianie ich prognoz. Zazwyczaj radzi sobie lepiej niż pojedyncze drzewo decyzyjne, ponieważ redukuje ryzyko przeuczenia i zwiększa dokładność prognoz.
2. **Gradient Boosting:** To kolejny model ensemble, który iteracyjnie trenuje drzewa decyzyjne w taki sposób, że każde kolejne drzewo stara się poprawić błędy popełnione przez poprzednie. Jest często uważany za jeden z najbardziej efektywnych dostępnych algorytmów uczenia maszynowego dla zadań regresyjnych i klasycyfikacyjnych.

W typowych scenariuszach, zarówno Random Forest, jak i Gradient Boosting, mają potencjał do osiągnięcia lepszych wyników niż prosta regresja liniowa, zwłaszcza w przypadkach, gdy zależności w danych są bardziej złożone i nieliniiowe. Jednakże wymagają one również bardziej starannej regulacji parametrów i mogą być bardziej podatne na przeuczenie, szczególnie w przypadku niewielkich zestawów danych.

Jeśli zdecydujesz się na użycie tych modeli, zalecam eksperymentowanie z różnymi ustawieniami parametrów, takimi jak liczba drzew w Random Forest czy prędkość uczenia w Gradient Boosting, aby znaleźć optymalną konfigurację dla Twoich danych.

python

Copy code

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# Training a Random Forest Regressor
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train)
y_pred_rf = rf_reg.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# Training a Gradient Boosting Regressor
gb_reg = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_reg.fit(X_train, y_train)
y_pred_gb = gb_reg.predict(X_test)
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

mse_rf, r2_rf, mse_gb, r2_gb
```

# SYSTEM REKOMENDACJI FILMÓW NETFLIX



# PROMPT STARTOWY



Ja:

Cześć, chciałbym, abyś wytrenował model uczenia nienadzorowanego do rekommendowania filmów, wykorzystując dane filmów Netflix'a. Dane znajdują się w załączniku.



netflix\_titles.csv  
Spreadsheet



ChatGPT:

Świetnie! Zaczniemy od wytrenowania modelu uczenia nienadzorowanego do rekommendacji filmów na podstawie danych z Netflix. Najpierw przyjrzymy się strukturze dostarczonego pliku CSV, aby zrozumieć, jakie informacje zawiera i jak możemy je wykorzystać. Następnie omówimy możliwe podejścia do budowy systemu rekommendacji. Przejdę teraz do analizy pliku.

python

Copy code

```
import pandas as pd

# Load the dataset
file_path = '/mnt/data/netflix_titles.csv'
netflix_data = pd.read_csv(file_path)

# Display the first few rows of the dataset to understand its structure
netflix_data.head()
```

Plik danych z Netflix zawiera następujące kolumny:

- `show\_id`: Unikalny identyfikator programu.
- `type`: Typ programu (film lub serial telewizyjny).
- `title`: Tytuł programu.
- `director`: Reżyser.
- `cast`: Obsada.
- `country`: Kraj produkcji.
- `date\_added`: Data dodania do Netflix.
- `release\_year`: Rok wydania.
- `rating`: Ocena wiekowa.
- `duration`: Czas trwania.
- `listed\_in`: Kategorie, do których program jest zaliczany.
- `description`: Opis programu.

Do stworzenia systemu rekommendacji<sup>[16]</sup>, możemy skorzystać z różnych podejść. Jednym z nich jest wykorzystanie informacji tekstowych, takich jak opis, tytuł, obsada, i kategoria, do stworzenia cech (ang. features), które będą podstawą do znalezienia podobieństw między filmami. Możemy użyć technik przetwarzania języka naturalnego (NLP<sup>[17]</sup>) do przekształcenia tych danych tekstowych w wektory, a następnie zastosować algorytmy takie jak K-means lub podobieństwo cosinusowe do grupowania lub porównywania filmów.

Przed przystąpieniem do modelowania, warto również przeprowadzić wstępную obróbkę danych, taką jak usunięcie brakujących wartości, zamiana tekstów na małe litery, usunięcie znaków interpunkcyjnych i podobnych.

Chciałbym, abyś poszedł dalej i zaczął przygotowywać dane, a następnie wybrał odpowiednią metodę do budowy systemu rekommendacji?

# KONTYNUACJA



Ja:

Tak. Jeśli napotkasz jakiś problem techniczny podaj mi kod abym mógł zastosować go w swoim środowisku.



ChatGPT:

W porządku, zaczniemy od przygotowania danych. Skoncentrujemy się na kilku kolumnach, które mogą być najbardziej pomocne w budowaniu systemu rekommendacji: `'title'`, `'cast'`, `'country'`, `'listed_in'` (kategorie), i `'description'`. Wykonamy następujące kroki:

1. **Czyszczenie danych:** Usuniemy brakujące wartości i przetworzymy tekst.
2. **Przetwarzanie tekstu:** Zamienimy tekst na małe litery, usuniemy znaki interpunkcyjne i podzielimy tekst na słowa (tokenizacja).
3. **Wektoryzacja tekstu:** Użyjemy metody, takiej jak TF-IDF<sup>[18]</sup> (Term Frequency-Inverse Document Frequency), aby przekształcić tekst w wektory liczbowe.
4. **Budowa modelu rekommendacji:** Możemy użyć metody podobieństwa cosinusowego lub innych technik uczenia maszynowego do oceny podobieństwa między filmami.

Zaczniemy od kroków 1 i 2, czyli od czyszczenia i przetwarzania tekstu. Po zakończeniu tego etapu, możemy przejść do wektoryzacji tekstu i budowy modelu rekommendacji. Przygotuję teraz odpowiedni kod.

python

```
# Vectorization using TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(netflix_data['combined_features'])

# Example: Displaying the shape of the TF-IDF matrix
tfidf_matrix.shape
```

Copy code

Wektoryzacja TF-IDF została zakończona, co dało nam macierz o wymiarach 8807 x 53785. Oznacza to, że mamy 8807 filmów reprezentowanych za pomocą 53785 unikalnych cech (słów).

Teraz możemy użyć tej macierzy do znalezienia podobieństw między filmami. Użyjemy podobieństwa cosinusowego, które jest popularną miarą w systemach rekommendacji do oceny podobieństwa między różnymi dokumentami (w naszym przypadku filmami).

Założymy, że chcesz znaleźć filmy podobne do konkretnego filmu. Wybierzmy na przykład film z indeksem 0 (pierwszy film w naszym zbiorze danych) i znajdźmy filmy, które są do niego najbardziej podobne. Poniżej przedstawię kod, który to umożliwia. Następnie możesz zastosować ten sam schemat w swoim środowisku, wybierając inne filmy jako punkty odniesienia.

python

Copy code

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re

# Data Preprocessing
# Filling missing values and combining text columns
netflix_data.fillna('', inplace=True)
netflix_data['combined_features'] = netflix_data['title'] + ' ' + netflix_data['cast']

# Function to clean and preprocess text
def clean_text(text):
    text = text.lower() # convert to lowercase
    text = re.sub(r'\s+', ' ', text) # replace multiple spaces with single space
    text = re.sub(r'[^w\s]', '', text) # remove punctuation
    return text

# Applying the cleaning function to combined features
netflix_data['combined_features'] = netflix_data['combined_features'].apply(clean_text)

# Display the cleaned combined features of the first few rows
netflix_data['combined_features'].head()
```

Dla filmu "Dick Johnson Is Dead" (pierwszy film w naszym zbiorze danych), model rekommendacyjny na podstawie podobieństwa cosinusowego zaproponował następujące filmy jako najbardziej podobne:

1. "The Death and Life of Marsha P. Johnson"
2. "End Game"
3. "How to Be a Player"
4. "New Girl"
5. "Win It All"

Możesz użyć tej metody, aby znaleźć rekommendacje dla innych filmów w zbiorze danych. Aby to zrobić, wystarczy podać tytuł filmu jako argument funkcji `find\_similar\_movies`. Poniżej znajdziesz kod, który możesz wykorzystać w swoim środowisku:

python

Copy code

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re

# Wczytywanie danych
netflix_data = pd.read_csv('ściezka_do_pliku_csv') # Podmień na odpowiednią ścieżkę do pliku CSV

# Przetwarzanie i czyszczenie danych
netflix_data.fillna('', inplace=True)
netflix_data['combined_features'] = netflix_data['title'] + ' ' + netflix_data['cast']
netflix_data['combined_features'] = netflix_data['combined_features'].apply(lambda x: x.strip())

# Wektoryzacja TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(netflix_data['combined_features'])

# Funkcje pomocnicze
def get_title_from_index(index):
    return netflix_data[netflix_data.index == index]['title'].values[0]

def get_index_from_title(title):
    return netflix_data[netflix_data.title == title].index.values[0]

def find_similar_movies(movie_title, num_recommendations=5):
    movie_index = get_index_from_title(movie_title)
    cosine_similarities = cosine_similarity(tfidf_matrix[movie_index], tfidf_matrix)
    similar_movies = list(enumerate(cosine_similarities[0]))
    sorted_similar_movies = sorted(similar_movies, key=lambda x: x[1], reverse=True)
    recommended_movies = []
    for i in sorted_similar_movies:
        movie_title = get_title_from_index(i[0])
        recommended_movies.append(movie_title)
    return recommended_movies

# Przykład użycia
test_movie_title = 'Podaj tytuł filmu'
recommended_movies = find_similar_movies(test_movie_title)
print(recommended_movies)
```

# REKOMENDACJE DLA TERMINATORA 3



Ja:

Czy, korzystając ze stworzonego przez ciebie systemu rekommendacji, możesz polecić mi inne filmy podobne do "Terminator 3: Rise of the Machines"?



ChatGPT:

Filmy rekommendowane jako podobne do "Terminator 3: Rise of the Machines" według stworzonego systemu rekommendacji to:

1. "Terminator Salvation"
2. "Fat Ballerina - David A. Arnolda"
3. "9"
4. "Total Recall"
5. "Stardust"

Mam nadzieję, że te rekommendacje będą dla Ciebie interesujące!

Kod:

```
python
```

```
# Using the recommendation system to find movies similar to "Terminator 3: Rise of the  
recommended_movies_for_terminator_3 = find_similar_movies("Terminator 3: Rise of the  
recommended_movies_for_terminator_3
```

Copy code



# OCENA REKOMENDACJI I SUGESTIE ULEPSZEŃ SYSTEMU



Ja:

Twoim zdaniem, czy podane przez system filmy są dobrą rekomendacją? Co można by zrobić aby poprawić przedstawiony przez ciebie system rekomendacji.



ChatGPT:

Ocena jakości rekomendacji zależy od wielu czynników, w tym od osobistych preferencji widza i kontekstu wyboru filmów. System rekomendacji oparty na podobieństwie cosinusowym i wektoryzacji TF-IDF, który zaimplementowałem, jest podstawowym podejściem i ma swoje ograniczenia. Oto kilka uwag na temat rekomendacji i sugestii dotyczących ulepszeń:

## Ocena Rekomendacji

- Znaczenie Kontekstowe:** Filmy takie jak "Terminator Salvation" i "Total Recall" wydają się być dobrymi rekomendacjami ze względu na podobny gatunek (science fiction, akcja). Jednakże inne tytuły, jak "Fat Ballerina - David A. Arnold", mogą wydawać się mniej trafne, co sugeruje, że algorytm mógł skupić się na innych cechach niż te, które są najważniejsze dla konkretnego gatunku lub stylu filmu.

## Sposoby Poprawy Systemu

- Lepsze Przetwarzanie Tekstowe:** Udoskonalenie przetwarzania tekstu, w tym lepsza tokenizacja i lematyzacja<sup>[23]</sup>, mogłoby poprawić jakość wektoryzacji.
- Zastosowanie Metod NLP:** Użycie bardziej zaawansowanych metod NLP, takich jak osadzenia słów (word embeddings<sup>[24]</sup>) z wykorzystaniem modeli takich jak Word2Vec lub BERT, mogłoby poprawić sposób reprezentacji znaczeń w opisach filmów.
- Uwzględnienie Oceny Użytkowników:** Integracja ocen użytkowników mogłaby pomóc w dostosowaniu rekomendacji do indywidualnych preferencji.
- Klasteryzacja Gatunków:** Wykorzystanie klasteryzacji do identyfikacji i rekomendacji filmów wewnętrz konkretnych gatunków mogłoby zwiększyć trafność sugestii.
- Uczenie Maszynowe i Głębokie Uczenie:** Zastosowanie zaawansowanych technik uczenia maszynowego i głębokiego uczenia, takich jak sieci neuronowe, mogłoby znaczco poprawić dokładność systemu rekomendacji.
- Feedback Użytkowników:** Włączenie mechanizmu feedbacku, który pozwala użytkownikom oceniać rekomendacje, może pomóc w ciągłym ulepszaniu systemu.
- Hybrydowy System Rekomendacji:** Połączenie kilku podejść, takich jak rekomendacje oparte na treści i filtrowanie kolaboratywne, może dać bardziej zróżnicowane i trafne wyniki.

Wdrożenie tych usprawnień może znaczco zwiększyć skuteczność systemu rekomendacji. Warto jednak pamiętać, że każdy system rekomendacji będzie miał swoje ograniczenia i nie zawsze będzie w stanie idealnie dopasować się do indywidualnych preferencji każdego użytkownika.

# TRENOWANIE AGENTA W ŚRODOWISUKU KUNG-FU MASTER



# PROMPT STARTOWY



Ja:

Witaj, Chat GPT. Zainteresowałem się środowiskiem OpenAI Gym o nazwie KungFuMaster i chciałbym stworzyć dla niego agenta AI. Móglbyś mi wyjaśnić, jak pobrać to środowisko do Pythona i następnie jak rozpocząć proces uczenia agenta gry?



ChatGPT:

Oczywiście, mogę Ci w tym pomóc. OpenAI Gym to biblioteka, która dostarcza szeroki zbiór środowisk testowych, które pomagają w rozwijaniu i porównywaniu algorytmów uczenia ze wzmacnianiem. Oto kroki, które musisz wykonać, aby rozpocząć pracę ze środowiskiem KungFuMaster w OpenAI Gym:

1. **Instalacja OpenAI Gym:**

Musisz zainstalować pakiet OpenAI Gym. Możesz to zrobić za pomocą pip, narzędzia do instalacji pakietów dla Pythona. Uruchom następujące polecenie w terminalu lub wierszu poleceń:

```
python
```

Copy code

```
pip install gym
```

2. **Instalacja środowisk Atari:**

Ponieważ KungFuMaster jest częścią środowisk Atari w OpenAI Gym, musisz również zainstalować gym[atari]. Możesz to zrobić następującym poleceniem:

```
python
```

Copy code

```
pip install gym[atari]
```

3. **Importowanie Gym i inicjalizacja środowiska:**

Po zainstalowaniu pakietów możesz zaimportować Gym do swojego skryptu Pythona i zainicjować środowisko KungFuMaster:

```
python
```

Copy code

```
import gym  
env = gym.make('KungFuMaster-v0')
```

#### 4. Rozpoczęcie procesu uczenia:

Aby rozpocząć proces uczenia, musisz zaimplementować algorytm uczenia ze wzmocnieniem, który będzie interaktywał ze środowiskiem. Oto bardzo prosty przykład, w którym agent podejmuje losowe działania:

```
python
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        action = env.action_space.sample() # Wybierz losowe działanie
        observation, reward, done, info = env.step(action)
        if done:
            print("Epizod zakończony po {} krokach".format(t+1))
            break
env.close()
```

#### 5. Implementacja algorytmu uczenia ze wzmocnieniem:

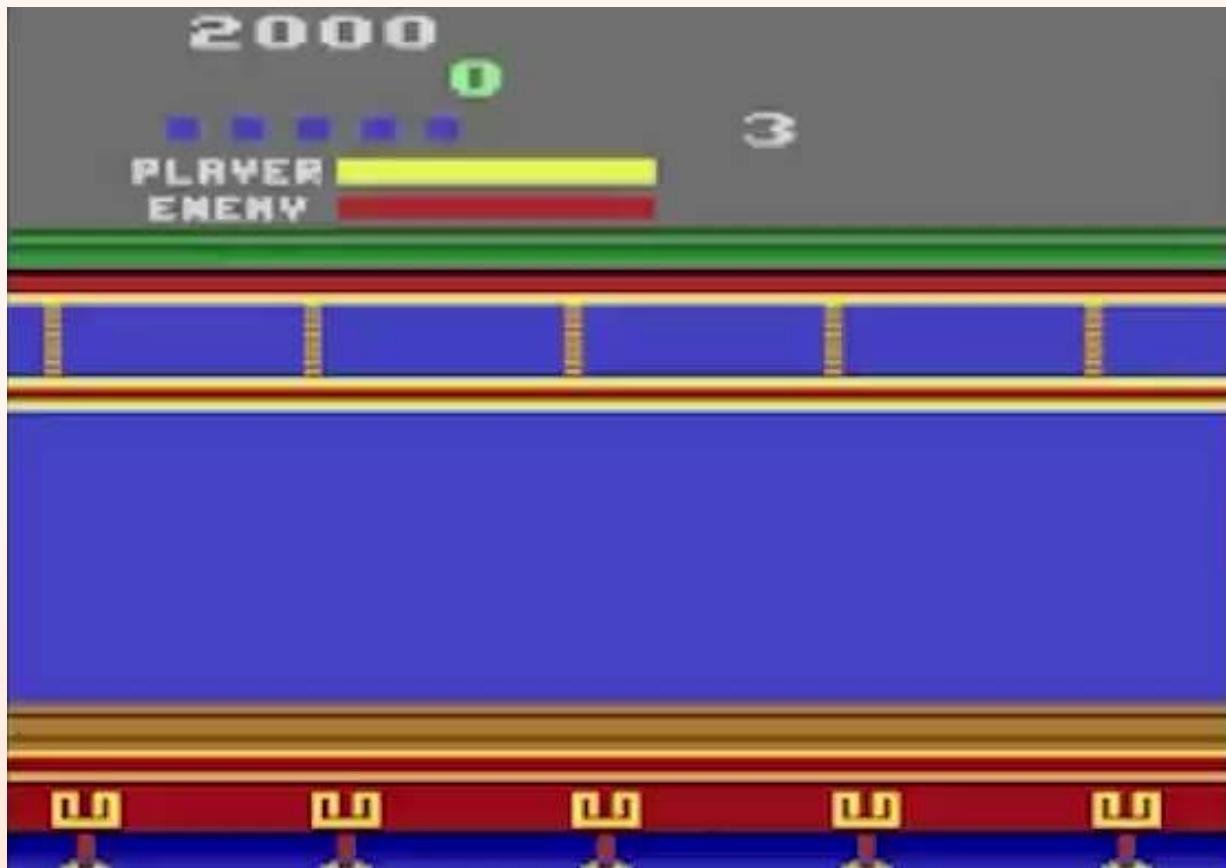
Proces uczenia wymaga bardziej skomplikowanego podejścia, takiego jak Q-learning, Deep Q Network (DQN), Policy Gradients, itp. Musisz zaimplementować taki algorytm, który będzie mógł nauczyć się strategii maksymalizującej nagrody w grze.

#### 6. Trenowanie i ocena agenta:

Po zaimplementowaniu algorytmu będziesz musiał trenować swojego agenta na wielu epizodach gry, monitorując jego postępy i dostosowując parametry algorytmu w razie potrzeby.

To jest tylko bardzo ogólne wprowadzenie do procesu. Każdy z tych kroków wymaga głębszego zrozumienia szczegółów i potencjalnie znacznej ilości kodu, zwłaszcza jeśli chodzi o implementację i dostrojenie algorytmów uczenia ze wzmocnieniem. Jeśli potrzebujesz bardziej szczegółowych informacji na każdym z tych etapów, daj mi znać!

# LOSOWE AKCJE



# TRENOWANIE AGENTA



Ja:

Skorzystałem z twojego kodu i zostało wyświetlane środowisko w którym agent wykonywał losowe działania. Co muszę teraz zrobić aby wytrenować agenta na tym środowisku?



ChatGPT:

Wytrenowanie agenta w środowisku KungFuMaster z OpenAI Gym wymaga zaimplementowania algorytmu uczenia ze wzmacnieniem. Istnieje wiele różnych algorytmów, które możesz użyć, a ich wybór zależy od wielu czynników, takich jak złożoność środowiska, wymagania dotyczące wydajności, dostępne zasoby obliczeniowe itp. Oto kroki, które musisz wykonać, aby wytrenować agenta:

## 1. Wybierz algorytm uczenia ze wzmacnieniem:

- *Q-Learning: Dla prostych problemów.*
- *Deep Q-Network (DQN)<sup>[25]</sup>: Łączy Q-learning z głębokimi sieciami neuronowymi.*
- *Policy Gradients<sup>[26]</sup>: Bezpośrednie uczenie polityk działania.*
- *Actor-Critic<sup>[27]</sup>: Kombinuje zalety Q-learningu i gradientów polityk.*
- *Proximal Policy Optimization (PPO) lub Trust Region Policy Optimization (TRPO): Dla bardziej zaawansowanych zastosowań.*

## 2. Zaprojektuj sieć neuronową:

Jeśli wybrałeś algorytm oparty na sieciach neuronowych, takie jak DQN lub PPO, będziesz musiał zbudować sieć neuronową, która będzie służyła jako aproksymator funkcji wartości lub polityki. W Pythonie zazwyczaj używa się do tego biblioteki TensorFlow lub PyTorch.

## 3. Zbieraj doświadczenie:

Interakcja z środowiskiem jest kluczowa. Agent musi wykonywać działania, obserwować nagrody i nowe stany, a następnie używać tych informacji do aktualizacji swojej polityki.

Oto przykładowy szkielet kodu, który pokazuje, jak można zainicjować proces uczenia (zakładając, że korzystamy z metody DQN):

python

```
import gym
from stable_baselines3 import DQN

# Inicjalizacja środowiska
env = gym.make('ALE/KungFuMaster-v5')

# Inicjalizacja agenta DQN
model = DQN('CnnPolicy', env, verbose=1)

# Trenowanie agenta
model.learn(total_timesteps=10000)

# Testowanie agenta
obs = env.reset()
for i in range(1000):
    action, _states = model.predict(obs, deterministic=True)
    obs, rewards, done, info = env.step(action)
    env.render()
    if done:
        break
env.close()
```

# BŁĄD “MEMORY ERROR”



Ja:

Podczas użycia twojego kodu do trenowania agenta pojawił się ten błąd.

MemoryError: Unable to allocate 93.9 GiB for an array with shape (1000000, 1, 3, 210, 160) and data type uint8



ChatGPT:

Błąd MemoryError wskazuje, że proces próbuje zaalokować więcej pamięci RAM, niż jest dostępne w systemie. W kontekście uczenia ze wzmacnianiem i używania sieci DQN, taki błąd jest często spowodowany przez próbę załadowania zbyt wielkiej ilości danych do pamięci, np. w przypadku pamięci powtórek (experience replay buffer).

Oto kilka kroków, które możesz podjąć, aby rozwiązać ten problem:

- Zmniejsz rozmiar bufora pamięci powtórek:** W DQN, bufor pamięci powtórek przechowuje poprzednie przejścia, które są wykorzystywane do uczenia sieci. Zmniejszenie liczby przechowywanych przejść może pomóc zmniejszyć zużycie pamięci.
- Użyj mniejszych wsadów (batch size):** Podczas treningu sieci, każda aktualizacja jest wykonywana na podstawie próbki (batch) z bufora pamięci powtórek. Mniejsze wsady zmniejszą wymagania pamięciowe.
- Zmniejsz częstotliwość aktualizacji:** Możesz również zaktualizować sieć rzadziej, co może zmniejszyć ilość pamięci potrzebnej na jednorazowe przechowywanie danych do aktualizacji.
- Zmniejsz rozmiar lub złożoność modelu:** Użycie prostszej sieci neuronowej z mniejszą liczbą parametrów może zmniejszyć ilość pamięci potrzebnej do przechowywania i aktualizacji sieci.
- Zmiana reprezentacji danych:** Jeśli używasz obrazów, możliwe, że przetwarzasz je w pełnej rozdzielczości. Możesz spróbować zmniejszyć rozdzielczość lub użyć innej metody przetwarzania wstępnego, aby zmniejszyć rozmiar danych wejściowych.
- Wykorzystanie pamięci dyskowej:** Jeśli żadne z powyższych rozwiązań nie jest możliwe, możesz skorzystać z technik takich jak pamięć wirtualna, przenosząc część danych z pamięci RAM na dysk twardy, co może być wolniejsze, ale pozwoli na przetwarzanie większych danych.

6. **Wykorzystanie pamięci dyskowej:** Jeśli żadne z powyższych rozwiązań nie jest możliwe, możesz skorzystać z technik takich jak pamięć wirtualna, przenosząc część danych z pamięci RAM na dysk twardy, co może być wolniejsze, ale pozwoli na przetwarzanie większych danych.

7. **Sprawdź wycieki pamięci:** Upewnij się, że nie masz wycieków pamięci w swoim kodzie, które mogłyby powodować, że z biegiem czasu zużycie pamięci stale rośnie.

8. **Upgrade sprzętu:** Jeśli żadne z powyższych rozwiązań nie jest wystarczające, możliwe, że będziesz musiał rozważyć ulepszenie swojego sprzętu komputerowego, aby miał więcej pamięci RAM.

Jeśli jesteś ograniczony sprzętowo, polecamy rozpoczęć od pierwszych czterech kroków, ponieważ są to zmiany, które możesz wprowadzić w kodzie bez potrzeby inwestowania w nowy sprzęt.

Oto jak można zmodyfikować rozmiar bufora i rozmiar partii w kodzie:

python

Copy code

```
from stable_baselines3.common.buffers import ReplayBuffer

buffer_size = 10000 # Zmniejsz tę liczbę, jeśli zabraknie ci pamięci
batch_size = 32 # Zmniejsz tę liczbę, aby zmniejszyć zużycie pamięci na partię

model = DQN('MlpPolicy', env, buffer_size=buffer_size, batch_size=batch_size, verbose=1)
```

Pamiętaj, że zmniejszenie rozmiaru bufora i wielkości partii może wpłynąć na uczenie się agenta, więc musisz znaleźć równowagę między zużyciem pamięci a wydajnością algorytmu.

# O C E N A W Y N I K Ó W N A J L E P S Z E G O A G E N T A



Ja:

Dziękuję. Korzystając z kodów wcześniej przez ciebie dostarczonych wytrenowałem finalnego agenta. Jednakże tym razem wykorzystałem najlepsze hiperparametry znalezione przez bibliotekę optuna. Wykonałem to w taki sposób:

```
python
# Wyciągnięcie najlepszych parametrów z obiektu 'study'
best_params = study.best_trial.params

# Stworzenie środowiska, możesz użyć tego samego środowiska co wcześniej
env = make_atari_env('ALE/KungFuMaster-v5', n_envs=1)
env = VecFrameStack(env, n_stack=4)

# Tworzenie modelu z najlepszymi hiperparametrami
model = DQN('CnnPolicy', env, buffer_size=10_000, batch_size=32,
            **best_params,
            tensorboard_log="..tensorboard_logs/",
            verbose=1)

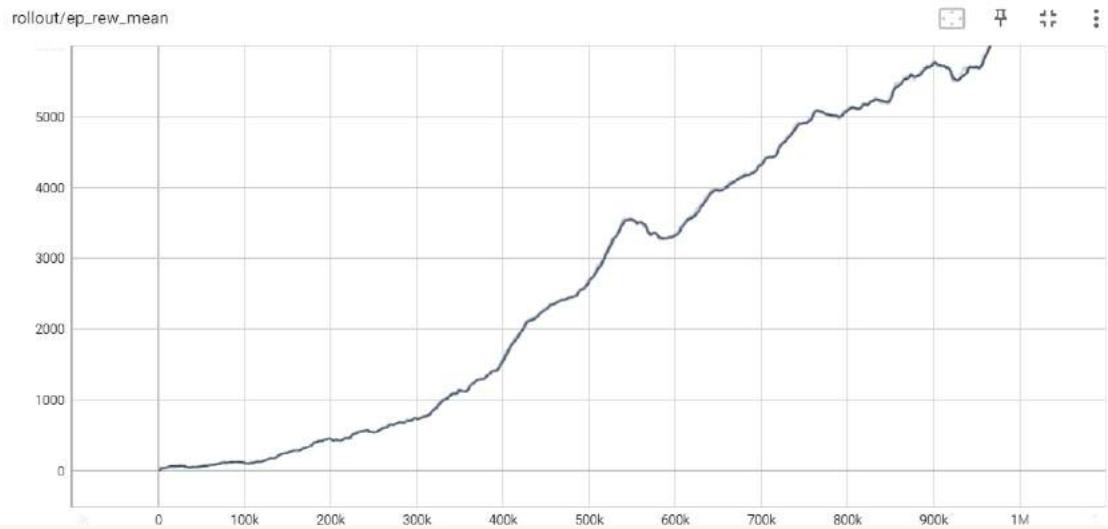
# Trenowanie modelu
model.learn(total_timesteps=1_000_000, log_interval=4);
```

Dzięki bibliotece TensorBoard uzyskałem wykresy, które pokazują osiągnięcia agenta w trakcie treningu. Wykresy te prześlę ci w załączniku abyś mógł je skomentować.

Co do moich spostrzeżeń/obserwacji gry wytrenowanego agenta, stwierdzam że radzi on sobie o wiele lepiej niż poprzednio wytrenowany model. Przeciwnicy specjalizujący się w walce bezpośredniej, którzy zbliżają się do gracza, są łatwo eliminowani przez model, a przeciwnicy rzucający szurikenami, szczególnie na poziomie głowy, są z łatwością unikani przez agenta. Natomiast szurikeny rzucane na poziomie nóg często trafiają w cel, co sugeruje, że agent nie nauczył się jeszcze efektywnie unikać ataków skierowanych na niższe partie ciała.



rollout/ep\_rew\_mean



rollout/exploration\_rate



NAJLEPSZY AGENT





Gratuluję postępów z Twoim agentem! Na podstawie przesłanych wykresów TensorBoard, można zauważać następujące informacje:

1. **Średnia nagroda za epizod (ep\_rew\_mean):** Wykres pokazuje stabilny wzrost średniej nagrody, co wskazuje na to, że Twój agent uczy się i poprawia swoje działania w środowisku. Regularność wzrostu świadczy o tym, że proces uczenia jest stabilny i skuteczny.
2. **Średnia długość epizodu (ep\_len\_mean):** Analogicznie do średniej nagrody, średnia długość epizodu również wzrasta, co wskazuje, że agent przetrwał dłużej w grze. Jest to kolejny dobry znak, sugerujący, że agent staje się lepszy w unikaniu końca gry (np. unikaniu przeciwników, trudnych sytuacji).
3. **Wskaźnik eksploracji (exploration\_rate):** Ten wykres pokazuje, jak zmniejsza się eksploracja wraz z postępami w uczeniu. Początkowo wysoki wskaźnik eksploracji jest pożądany, aby agent mógł nauczyć się różnorodnych strategii. Z czasem jednak eksploracja maleje, co pozwala agentowi wykorzystywać nauczone już strategie do maksymalizacji nagród.

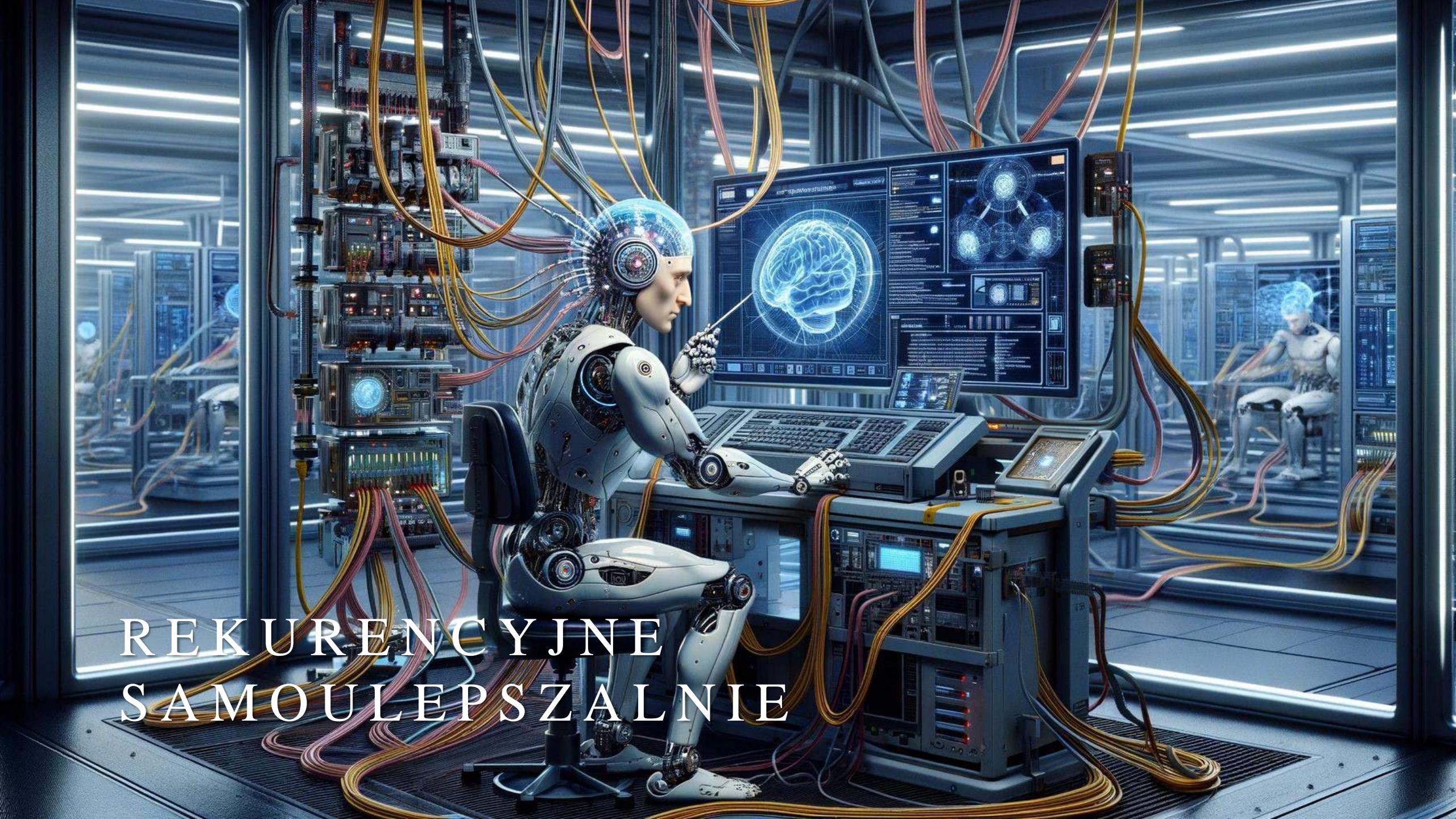
Twoje obserwacje dotyczące wydajności agenta są bardzo cenne i sugerują, że agent nauczył się niektórych kluczowych aspektów gry, takich jak walka bliska i unikanie. To, że agent nie unika skutecznie szurikenów rzuconych na poziomie nóg, może wskazywać na konieczność dalszego uczenia lub potrzebę wprowadzenia specyficznych nagród kształtujących, które zachęcałyby do unikania ataków nisko celowanych.

Możliwe, że dodatkowe dostosowanie funkcji nagrody lub **dalsze trenowanie z większą liczbą epizodów** mogłyby pomóc agentowi lepiej rozumieć i reagować na te niższe ataki. Można też rozważyć modyfikację architektury sieci neuronowej, aby lepiej przetwarzała informacje wizualne dotyczące dolnych regionów ekranu, gdzie pojawiają się te szurikeny.

Jeśli masz możliwość przekazywania bardziej szczegółowych informacji zwrotnych agentowi, może to także pomóc w nauce unikania ataków nisko celowanych. Inną strategią może być zastosowanie uczenia przez demonstrację (*learning from demonstration*), gdzie agent mógłby uczyć się z gry eksperta ludzkiego, który efektywnie unika takich ataków.

# PODSUMOWANIE

- Jak ChatGPT-4 sprawował się w uczeniu maszynowym?
- Czego można się spodziewać po kolejnych wersjach ChatGPT i modelach językowych?



REKURENCYJNE  
SAMOULEPSZALNIE

THANK YOU  
FOR  
YOUR ATTENTION

ARTIFICIAL INTELLIGENCE