Zadanie rekrutacyjne do KN Solvro

Sekcja uczenia maszynowego

wiosna 2025

Krzysztof Głowacz

Spis treści

1 Wstęp

W niniejszym raporcie przedstawione zostało rozwiązanie zadania rekrutacyjnego do sekcji uczenia maszynowego Koła Naukowego Solvro (rekrutacja wiosenna 2025). Zadanie polegało na eksploracyjnej analizie danych oraz klasteryzacji podanego zbioru [1].

2 Analiza danych

Zbiór danych, pochodzący z bazy danych TheCocktailDB, zawierał listę koktajli wraz ze składnikami niezbędnymi do ich przyrządzenia. Link do zbioru danych został dołączony do sekcji Źródeł tego raportu [2].

W celu przeprowadzenia wstępnej analizy danych (*EDA*) napisany został skrypt `src/eda.py`, który miał za zadanie:

- wczytać dane z pliku w formacie JSON do formatu DataFrame z biblioteki Pandas,
- wyświetlić podstawowe statystyki danych,
- wygenerować pełny raport opisujący dane korzystając z biblioteki ydata-profiling.

Na zdjęciu nr 1 przedstawiony został fragment wyjścia standardowego po uruchomieniu skryptu.

```
❯ uv run src/eda.py
Upgrade to ydata-sdk
Improve your data and profiling with ydata-sdk, featuring data quality scoring, redundancy detection, outlier identification, text validation, and synthetic data generation.
Register at ttps://ydata.ai/register
2025-03-22 18:58:07 [info     ] Loading data from /home/kris/Studia/inne/kn-solvro/rekrutacja-2025/data/cocktail_dataset.json
2025-03-22 18:58:07 [info     ] Loaded dataset with shape: (134, 11)
2025-03-22 18:58:07 [info     ] Starting dataset exploration


=== Dataset Exploration ===
Number of cocktails: 134

Columns info:
    Column        Data Type      Missing Values    Missing Percentage
--  ------------  -----------  ----------------  --------------------
 0  id            int64                       0                     0
 1  name          object                      0                     0
 2  category      object                      0                     0
 3  glass         object                      0                     0
 4  tags          object                     99                 73.88
 5  instructions  object                      0                     0
 6  imageUrl      object                      0                     0
 7  alcoholic     int64                       0                     0
 8  createdAt     object                      0                     0
 9  updatedAt     object                      0                     0
10  ingredients   object                      0                     0

Category distribution
---------------------
category
Ordinary Drink         127
Cocktail                 6
Punch / Party Drink      1
Name: count, dtype: int64
```

Zdjęcie 1: Uruchomienie skryptu src/eda.py

3 Przygotowanie danych

Przedmiotem analizy był zbiór Jester [**?**], który obejmował 100 różnych żartów wraz z ocenami ich śmieszności od niespełna 75 tysięcy osób. Żarty oceniane były w skali $(-10.0, 10.0)$. Przygotowanie danych wymagało w pierwszej części połączenia 3 plików z ocenami (po ok. 24 tys. ocen na każdy z plików), a następnie dla każdego wiersza wybranie wszystkich kolumn poza pierwszą (gdyż ta zawierała informację o liczbie ocenionych żartów przez danego użytkownika) i zamianę wartości *99* na `NaN`, ponieważ oryginalny zbiór danych liczbą 99 sygnalizował brak oceny (*null*), którego reprezentacją w bibliotece **numpy** jest właśnie `NaN`. Ostatecznie zbiór wszystkich ocen miał rozmiar: 73421 wierszy $\times$ 100 kolumn i prezentował się następująco (pierwsze 10. wierszy, kolumny symbolizują numery kolejnych żartów):
Aby móc trenować model sieci neuronowej na zebranych danych konieczne było przypisanie jednej wartości dla każdego żartu, ponieważ celem modelu jest znalezienie takiej **funkcji**, która możliwie najlepiej opisze mapowanie treści żartu na poziom jego śmieszności. Funkcja natomiast z definicji jest takim przyporządkowaniem, które każdemu elementowi z jednego zbioru przyporządkowuje dokładnie jeden element z drugiego zbioru. Z tego powodu każdemu ze 100 żartów przypisana została średnia arytmetyczna z jego ocen. Otrzymano więc wektor długości 100, gdzie najmniejsza średnia ocen wyniosła: -3.705, a największa: 3.363. Wektor ten stał się wektorem etykiet `Y`.

	1	2	3	4	...	99	100
0	-7.82	8.79	-9.66	-8.16		NaN	NaN
1	4.08	-0.29	6.36	4.37		-4.32	1.07
2	NaN	NaN	NaN	NaN		NaN	NaN
3	NaN	8.35	NaN	NaN		NaN	NaN
4	8.50	4.61	-4.17	-5.39		1.80	1.60
5	-6.17	-3.54	0.44	-8.50		-5.05	-3.45
6	NaN	NaN	NaN	NaN		NaN	NaN
7	6.84	3.16	9.17	-6.21		1.31	0.00
8	-3.79	-3.54	-9.42	-6.89		-3.40	-4.95
9	3.01	5.15	5.15	3.01		NaN	NaN

Tabela 1: Przykładowe oceny śmieszności kolejnych żartów (od 10 osób)

W celu przygotowania wektora tekstów `X` połączono sto plików w formacie *html*, z których każdy zawierał tekst danego żartu. Przy użyciu biblioteki **beautifulsoup4** pobrane zostały treści żartów, które po dodatkowym przetworzeniu (usunięciu znaków białych, usunięciu prefiksów „Q." oraz „A." dla żartów z pytaniem i odpowiedzią, itp.) trafiły do jednej listy. Finalnie pierwsze pięć elementów listy wyglądało następująco:

1. A man visits the doctor. The doctor says "I have bad news for you.You have cancer and Alzheimer's disease". The man replies "Well, thank God I don't have cancer!"
2. This couple had an excellent relationship going until one day he came home from work to find his girlfriend packing. He asked her why she was leaving him and she...
3. What's 200 feet long and has 4 teeth? The front row at a Willie Nelson Concert.
4. What's the difference between a man and a toilet? A toilet doesn't follow you around after you use it.
5. What's O. J. Simpson's Internet address? Slash, slash, backslash, slash, slash, escape.

Tak przygotowane teksty żartów należało następnie poddać procesowi wektoryzacji opartemu o tzw. słownik. Każda treść żartu przekształcona została na wektor $x \in \mathbb{N}^D$, który na i-tej pozycji zawierał liczbę wystąpień słowa/sylaby o indeksie i. Podczas laboratorium przetestowano dwa modele przetwarzania języka naturalnego: **Bert (bert-base-cased)** oraz **FastText**. W przypadku tego pierwszego długość wektora x równa była 768, a w przypadku drugiego 300. Tak przekształcona lista żartów stała się wektorem `X`.

Ostatnim etapem przygotowania danych był podział wektorów `X` i `Y` na zbiór danych uczących i walidacyjnych. W tym celu skorzystano z funkcji `train_test_split` z biblioteki **sklearn.model_selection**. Ze względu na stosunkowo niewielki rozmiar zbioru danych wejściowych dokonano podziału w stosunku 3:7, tj. rozmiar zbioru danych testowych stanowił 30% rozmiaru zbioru wszystkich danych. Aby zapewnić powtarzalność całego procesu między kolejnymi jego wykonaniami wszystkie funkcje mogące zachowywać się w sposób pseudolosowy otrzymywały taki sam generator liczb pseudolosowych (z takim samym ziarnem).

4 Wykorzystanie modelu MLP

Multi-layer Perceptron (MLP) jest algorytmem uczenia nadzorowanego, który poszukuje funkcji

$$f : \mathbb{R}^m \to \mathbb{R}^o$$

(m - liczba wymiarów wejściowych, o - liczba wymiarów wyjściowych) poprzez trening na zbiorze danych uczących. Może zostać użyty w przypadku problemu regresji, a także klasyfikacji. Pomiędzy warstwami wejściową i wyjściową może znajdować się jedna lub więcej warstw, zwanych warstwami ukrytymi. Symboliczna reprezentacja sieci tego typu pochodząca z dokumentacji użytej biblioteki [**?**] zaprezentowana jest na poniższym zdjęciu:
W dalej opisanych eksperymentach używany był model `MLPRegressor`, którego najważniejsze hiperparametry (z punktu widzenia tego laboratorium) to:

Zdjęcie 2: Przykładowa sieć MLP z jedną warstwą ukrytą

1. **hidden_layer_sizes** - tablica, której i-ty element reprezentuje liczbę neuronów w i-tej warstwie ukrytej. We wszystkich testach używana była jedna warstwa ukryta. Domyślna wartość: *(100,)*.
2. **solver** - metoda optymalizacji wag. W testach używana była wartość *sgd*, która używa stochastyczny gradient do wyznaczenia kierunku poszukiwań. Domyślna wartość: *adam*.
3. **alpha** - siła regularyzacji L2. W testach używaną wartością było 0.0. Domyślna wartość: 0.0001.
4. **learning_rate** - strategia aktualizacji tempa uczenia. W testach ustawiona była wartość *constant*, która oznacza stałą prędkość uczenia parametryzowaną przez **learning_rate_init**. Domyślna wartość: *constant*.
5. **learning_rate_init** - początkowa prędkość uczenia, kontroluje wielkość „kroku" przy aktualizacji wag. Domyślna wartość: 0.001.
6. **random_state** - określa generator pseudolosowy używany przy niektórych wartościach, np. wagach. W testach używano `np.random.RandomState(42)` w celu zapewnienia powtarzalności między kolejnymi wykonaniami danego testu. Domyślna wartość: *None*.
7. **warm_start** - w przypadku ustawienia flagi `True` poprzednie rozwiązanie jest używane ponowne przy inicjalizacji następnego (w przeciwnym przypadku poprzednie rozwiązanie jest usuwane). W testach flaga była ustawiona na wartość `True`. Domyślna wartość: `False`.
8. **max_iter** - maksymalna liczba iteracji. W przypadku użycia metody *sgd* wartość ta określa liczbę epok. W testach ustawiona była wartość 1, ponieważ liczba epok kontrolowana była w inny sposób (przez zewnętrzny licznik), który umożliwiał łatwe tworzenie wykresów. Domyślna wartość: *200*.

4.1 Model MLP o domyślnej konfiguracji

Etap testowania wpływu poszczególnych hiperparametrów na jakość działania sieci neuronowej typu MLP rozpoczynał się od sprawdzenia działania podstawowego modelu o domyślnej konfiguracji. Kod odpowiedzialny za przygotowanie takiego podstawowego modelu wyglądał następująco:

```
rng = np.random.RandomState(42)

mlp = MLPRegressor(solver='sgd', alpha=0.0, learning_rate='constant', random_state=rng,
    warm_start=True, max_iter=1)
```

Wykresy na poniższych zdjęciach przedstawiają wartość funkcji kosztu (błąd średniokwadratowy) odpowiednio zbioru treningowego i walidacyjnego w zależności od epoki:
Wykres dla modelu Bert świadczy o tym, że po ok. 500 epokach pojawiło się zjawisko *overfittingu* - model MLP zaczął się zbyt dobrze dopasowywać do danych treningowych, na co wskazują rosnące wartości MSE dla zbioru walidacyjnego - model zaczął sobie gorzej radzić z predykowaniem wartości dla nieznanych próbek, ponieważ przeuczył się na danych treningowych. Wykres dla modelu FastText z kolei nie wykazuje oznak *overfittingu*, ale generalnie widoczne jest bardzo małe dopasowanie do danych w przypadku obu zbiorów, a drobne różnice między wartościami MSE dla zbioru uczącego i walidacyjnego mogą wynikać z natury samych zbiorów - dane testowe mogą zawierać przykładowo więcej szumów i trudniejszych tekstów do predykowania.

Źródła

[1] Pełna treść zadania rekrutacyjnego
`https://github.com/Solvro/rekrutacja/blob/main/machine_learning.md`.
Dostęp: 22.03.2025.

[2] Zbiór danych o koktajlach
`https://github.com/Solvro/rekrutacja/blob/main/data/cocktail_dataset.json`.
Dostęp: 22.03.2025.