Projekt SUML – Dokumentacja

Autorzy:

1. Paweł Wydrych (s25983)
2. Mikołaj Toczko (s28901)
3. Jakub Dominiczak (s27928)

Spis treści

1. Wprowadzenie

Celem projektu było stworzenie aplikacji „Kalkulator zdawalności”, która będzie na podstawie cech ucznia obliczać prawdopodobieństwo zdania egzaminu. Aplikacja korzysta z gotowego zbioru danych historycznych, które w pierwszej fazie projektu zostały poddane procesowi analizy oraz wstępnego przetwarzania. Następnie odpowiednio przetworzony zbiór danych został wykorzystany do wytrenowania modelu uczenia maszynowego, którego zadaniem jest wykrywanie zależności pomiędzy cechami ucznia a prawdopodobieństwem zdania egzaminu. Końcowym etapem projektu było udostępnienie wytrenowanego modelu jako aplikacji webowej, umożliwiającej wykonanie predykcji w prosty i intuicyjny dla użytkownika sposób. Dodatkowo istotnym aspektem projektu było zapewnienie, aby aplikacja była łatwo przenaszalna pomiędzy różnymi środowiskami. W tym celu cała aplikacja została skonteneryzowana przy użyciu platformy Docker. Zastosowanie tego rozwiązania uniezależnia działanie programu od systemu operacyjnego użytkownika oraz eliminuje konieczność ręcznej konfiguracji środowiska programistycznego i instalacji bibliotek. Dzięki temu aplikacja jest uniwersalna i gotowa do natychmiastowego uruchomienia na dowolnej maszynie.

2. Analiza i przygotowanie danych

Autor: Paweł Wydrych

2.1 Źródło danych

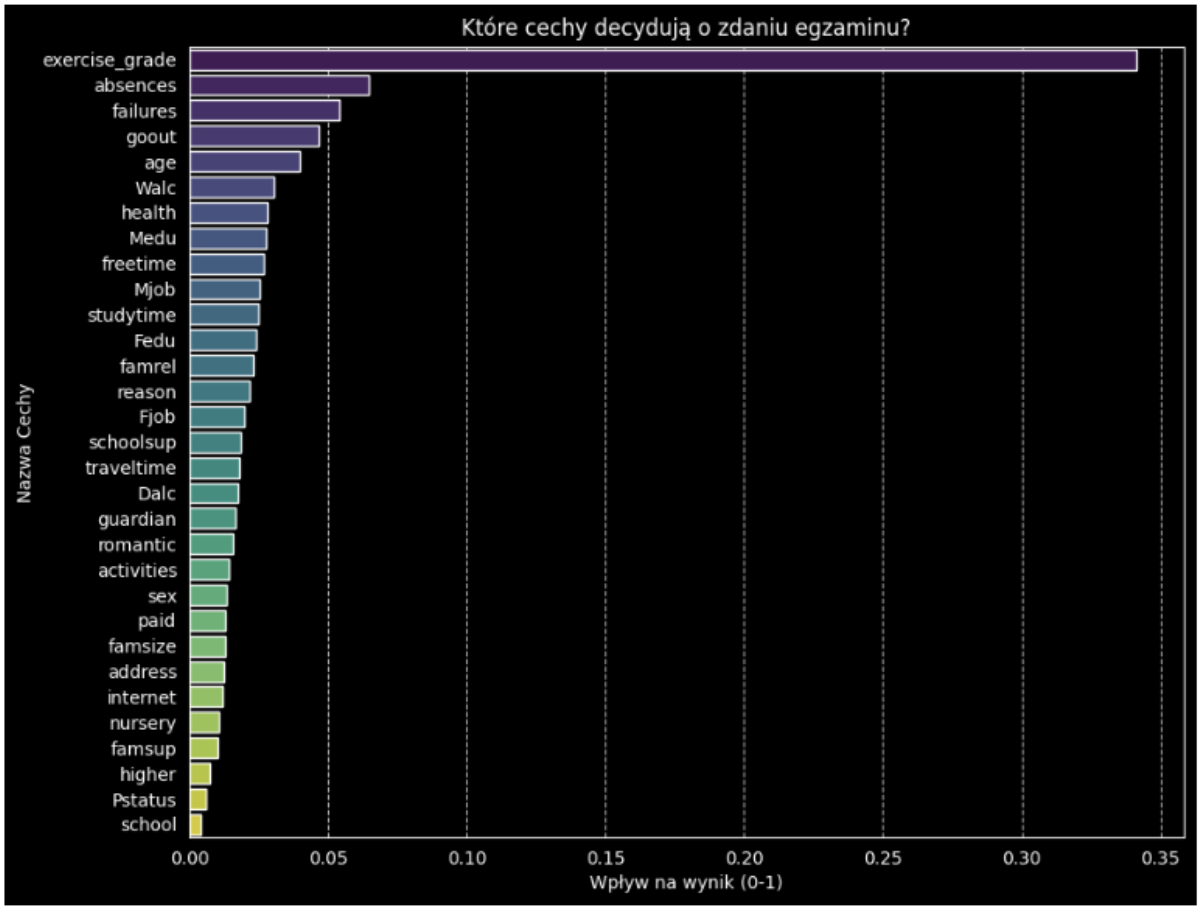
Jako podstawę do budowy modelu wykorzystano ogólnodostępny zbiór danych **"Student Performance Data Set"**, pochodzący z repozytorium UCI Machine Learning Repository. Zbiór ten zawiera dane dotyczące osiągnięć uczniów w edukacji średniej w dwóch portugalskich szkołach. Do celów projektu wybrano podzbiór dotyczący przedmiotu matematyka, który składa się z 395 rekordów oraz 33 atrybutów.

Atrybuty można podzielić trzy główne kategorie:

- **Informacje osobiste o uczniu** (np. wiek, płeć, wielkość rodziny).
- **Informacje o nauce w szkole** (np. czas nauki, liczba nieobecności, wcześniejsze niepowodzenia).
- **Zdrowie i styl życia** (np. spożycie alkoholu, czas wolny, stan zdrowia).

2.2 Selekcja cech

Oryginalny zbiór zawierał wiele zmiennych, które miały znikomy wpływ na wynik końcowy. Zdecydowano się odrzucić atrybuty nieistotne, takie jak: adres zamieszkania, zawód rodziców, powód wyboru szkoły czy imiona opiekunów. Skupiono się na zmiennych silnie skorelowanych z wynikiem, takich jak: **exercise_grade** (ocena początkowa), **failures** (liczba poprawek), **absences** (nieobecności) oraz **Walc/Dalc** (spożycie alkoholu).



Rysunek 1Feature importance datasetu (Opracowanie własne)

2.3 Przetwarzanie danych

Surowe dane wymagały odpowiedniego przygotowania, aby mogły zostać przetworzone przez algorytm uczenia maszynowego. Proces ten został zaimplementowany w pliku **process_data.py** i obejmował następujące kroki:

1. **Wczytanie zbioru danych**
   Wczytano zbiór danych oraz sprawdzono czy nie posiada on wartości pustych.

2. **Kodowanie zmiennych kategorycznych**
   Algorytmy ML operują na liczbach, dlatego zmienne tekstowe musiały zostać zamienione na wartości numeryczne.
   - Dla zmiennych binarnych (np. schoolsup, paid) zastosowano mapowanie: yes -> 1, no -> 0.
   - Dla zmiennych nominalnych (np. Mjob, Fjob) zastosowano technikę Label Encoding.

3. **Definicja targetu**
   Zmienną wynikową jest pass_exam (ocena końcowa w skali 0-20). Na potrzeby problemu klasyfikacji dokonano binaryzacji tej zmiennej:
   - wynik 1 (zdał) dla pass_exam >= 10
   - wynik 0 (nie zdał) dla pass_exam < 10

4. **Wybór kolumn istotnych do trenowania oraz odrzucenie pozostałych.**
   W procesie przygotowania danych ograniczono zbiór wejściowy do 14 kluczowych atrybutów, odrzucając cechy o niskim wpływie na wynik.

Wykorzystano następujące biblioteki:

1. **pandas** – wczytanie pliku CSV, mapowanie wartości tekstowych na liczbowe i wybór kolumn.
2. **numpy** – stworzenie binarnej zmiennej celu (pass_exam) przy użyciu warunku logicznego.
3. **os** – zarządzanie ścieżkami do plików.

3. Budowa i trenowanie modelu

Autor: Mikołaj Toczko

Proces trenowania modelu został zaimplementowany w pliku **train.py**.

3.1 Wybór algorytmu

Do klasyfikacji wykorzystano algorytm **Random Forest Classifier** z biblioteki scikit-learn.

3.2 Podział danych

Zbiór danych został podzielony na dwie części przy użyciu funkcji train_test_split:

- **Zbiór treningowy (80%):** Użyty do nauki wzorców.
- **Zbiór testowy (20%):** Użyty do weryfikacji skuteczności modelu na nieznanych danych.
- **Reprodukowalność:** Ustawiono parametr random_state, aby wyniki były powtarzalne przy każdym uruchomieniu.

3.3 Trenowanie modelu

- **Trenowanie:** Metoda .fit() dopasowała model do danych treningowych.
- **Serializacja:** Wytrenowany model został zapisany do pliku binarnego model.pkl przy użyciu biblioteki **Joblib**. Pozwala to na jego wielokrotne użycie w aplikacji bez konieczności ponownego trenowania.

3.4 Wyniki

Model oceniono na podstawie parametru **Accuracy.**

4. Implementacja aplikacji webowej

Autor: Jakub Dominiczak

Ostatnim etapem projektu było udostępnienie wytrenowanego modelu dla użytkowników w postaci aplikacji webowej.

4.1 Wybór technologii

Interfejs użytkownika został zrealizowany przy użyciu **Streamlit**. Jest to biblioteka języka Python, która umożliwia tworzenie interaktywnych aplikacji webowych bez konieczności pisania kodu w HTML, CSS czy JavaScript.

4.2 Integracja z modelem

1. **Mapowanie danych:** Aplikacja w locie tłumaczy wybory użytkownika (np. "Bardzo dobre zdrowie") na wartości liczbowe wymagane przez model (np. 5).

2. **Wczytanie modelu:** Przy starcie aplikacji biblioteka **joblib** ładuje plik model.pkl do pamięci.

3. **Predykcja:** Po kliknięciu przycisku **Oblicz szansę**, dane z formularza są pakowane w obiekt DataFrame, a model zwraca wynik klasyfikacji oraz prawdopodobieństwo (predict_proba).

4.3 Prezentacja wyników

Wynik predykcji jest prezentowany użytkownikowi w formie graficznej i tekstowej:

- **Sukces (Zda):** Komunikat w kolorze zielonym z informacją o stopniu pewności w procentach plus animacja.
- **Ryzyko (Nie zda):** Komunikat ostrzegawczy w kolorze czerwonym z informacją o stopniu pewności modelu.

5. Wymagania i instrukcja uruchomienia

5.1 Wymagania sprzętowe

Aplikacja została zaprojektowana z myślą o maksymalnej przenoszalności, jednak do jej poprawnego działania wymagane jest spełnienie podstawowych warunków:

- **Sprzęt:** Komputer z procesorem min. 2-rdzeniowym i 4GB pamięci RAM – wymagania Docker.
- **System operacyjny:** Dowolny system wspierający wirtualizację (Windows 10/11, macOS, Linux).
- **Oprogramowanie:** Zainstalowane środowisko **Docker** (wersja 4.0 lub nowsza).

5.2 Instrukcja uruchomienia

Zalecaną metodą uruchomienia jest wykorzystanie skonteneryzowanej wersji aplikacji, co gwarantuje izolację środowiska. Proces składa się z trzech kroków:

1. **Pobranie kodu:** Sklonowanie repozytorium projektu (polecenie git clone) z repozytorium:

   https://github.com/wydraaapw/SUML_student_exam_predictor

   ```
   git clone https://github.com/wydraaapw/SUML_student_exam_predictor.git
   cd SUML_student_exam_predictor
   ```

2. **Budowa obrazu:** Wykonanie komendy docker build, która tworzy obraz systemu na podstawie instrukcji zawartych w pliku Dockerfile.

   ```
   docker build -t exam-app .
   ```

3. **Uruchomienie kontenera:** Start aplikacji poleceniem docker run z przekierowaniem portu 8501. Aplikacja będzie dostępna pod url: http://127.0.0.1:8501/

```
docker run -p 8501:8501 exam-app
```

6. Podsumowanie

W ramach projektu powstała działająca aplikacja webowa, która pozwala przewidzieć szansę na zaliczenie egzaminu. Przeprowadzone testy na zbiorze walidacyjnym wykazały, że wytrenowany model osiąga dokładność (Accuracy) powyżej 80%. Jest to satysfakcjonujący wynik, który potwierdza, że dobrane przez nas cechy mają realny wpływ na sukces studenta. Aplikacja została poprawnie zintegrowana z Dockerem, co pozwala na jej bezproblemowe uruchomienie na każdym komputerze go obsługującym. Cel projektu został w pełni zrealizowany.