Modelowanie i analiza systemów informatycznych

dokumentacja projektu systemu ekspertowego do rozpoznawania cukrzycy wśród indian

Aleksander Kulpa, Mikołaj Macura, Paweł Habrzyk, grupa 2

10 grudnia 2023

Część I

Opis programu

Zaimplementuj system wspomagania lekarzy (poprzez użycie sieci neuronowej) poprzez automatyczną analizę danych medycznych. System posiada:

- dwa tryby – uczenie algorytmu/klasyfikacja nowej próbki
- dane medyczne szyfrowane i bezpieczne (o tym w następnej sekcji)
- dane lekarzy odpowiednio zabezpieczone (o tym w następnej sekcji)
- możliwość dodawania nowego pacjenta/lekarza
- wszystkie moduły zostały przetestowane
- sieć neuronowa została przeanalizowana pod względem ilości neuronów/warstw
- W zadaniu wykorzystaliśmy bazę danych: https://www.kaggle.com/uciml/pima-indians-diabetes-database

Instrukcja obsługi

Po pobraniu kodu źródłowego włączamy dwa terminale w ścieżce do kodu W jednym uruchamiamy komendę "npm run dev", która włączy frontend, w drugim uruchamiamy "pip install -r requirements.txt"a następnie "uvicorn app:app –reload".

0.1 Instrukcja wdrożenia

Dodatkowe informacje

Wymagania, podział pracy itd.

Część II

Opis działania

W tym skrypcie Pythona tworzona jest sieć neuronowa przy użyciu biblioteki `keras`. Sieć ta jest prosta, sekwencyjna, skonstruowana z kilku warstw.

Główna architektura sieci wygląda następująco:

1. Warstwa wejściowa: Warstwa `Dense` (gęstej) z 64 komórkami (neuronami) i funkcją aktywacji typu 'linear'. Ta warstwa przyjmuje dane wejściowe o określonym rozmiarze (rozmiarze cech wejściowych).

2. Druga warstwa: Warstwa robocza - `Dense` z 32 komórkami (neuronami) i funkcją aktywacji 'linear'.

3. Trzecia warstwa: Kolejna warstwa robocza - `Dense` z 16 komórkami (neuronami) i również z funkcją aktywacji 'linear'.

4. Warstwa wyjściowa: Końcowa warstwa - `Dense` z jednym neuronem i liniową funkcją aktywacji. Ta warstwa zwraca końcowy wynik prognozy.

Model sieci neuronowej jest trenowany przy użyciu optymalizatora 'adam' i funkcji straty jest 'mean_squared_error' (średni kwadrat błędu).

Natomiast w procesie tuningu hiperparametrów, stosowany jest algorytm `Random Search`. Wyszukuje on losowo kombinacje hiperparametrów, ocenia model dla każdej kombinacji i wybiera tę, która daje najmniejszą loss (stratę).

Podczas predykcji, model jest wczytywany z pliku, a następnie używany do prognozowania wartości wyjściowej na podstawie danych wejściowych. Prognoza jest następnie konwertowana do `int` i zwracana.

Wzory matematyczne:

- Adam: `https://arxiv.org/abs/1412.6980v8`
- Mean Squared Error:

$$MSE = \frac{1}{n}\sum(actual - prediction)^2$$

Algorytmy

Reprezentacja matematyczna modelu

$$y = W_4 \cdot (W_3 \cdot (W_2 \cdot (W_1 \cdot x + b_1) + b_2) + b_3) + b_4 \tag{1}$$

Biblioteki

- sklearn dla modelu regresji logistycznej
- keras dla modelu sekwencyjnego
- kerastuner do strojenia hiperparametrów.

Modelowanie

- Użycie Sequential z keras do tworzenia modelu sieci neuronowej.
- Konfiguracja warstw i neuronów w modelu (Dense layers).
- Kompilacja modelu z określonymi parametrami (np. optimizer='adam', loss='mean_squared_error')

Przetwarzanie Danych

- Użycie pandas do manipulacji i analizy danych.
- Podział danych na zestawy treningowe i testowe (train_test_split).

Bazy danych

Opis bazy i tabel

W projekcie użyta została baza danych SQLite Baza danych zawiera 3 tabele:

- diabetes - zawiera dane do szkolenia modelu
- patients - zawiera informacje o pacjencie oraz jego historię predykcji
- doctors - zawiara listę kont wraz z zaszyfrowanymi hasłami algorytmem bcrypt

Wszystkie table (łącznie z nazwą tabeli i kolumn) są zaszyfrowane algorytmemt md5 używając funkcji hashującej SHA512 dzięki bibliotece SQLCipher

Operacje na bazie danych

- Wstawianie i pobieranie danych o cukrzycy.
- Rejestracja użytkowników i autentyfikacja.

Implementacja systemu

app.py

Tutaj uruchamiana jest aplikacja Flask. Prócz rozruchu w pliku zawarta jest definicja naszej sieci neuronowej oraz definijcje endpointów RestAPI.

db_manager.py

W tym miejscu znajduje się klasa odpowiedzialna za inicjalizację oraz obsługę bazy danych, podczas działania programu.

Inicjalizacja bazy danych:

Otwórz i wczytaj bazę danych pod podaną ścieżką
if *Podany plik nie istnieje lub nie jest bazą danych* **then**
| Stwórz nową bazę danych pod podaną ścieżką i ją wczytaj
end
if *Baza danych nie zawiera, którejś z tabel "diabetes", "doctors"oraz "patients"* **then**
| Stwórz tabele "diabetes", "doctors"oraz "patients"zgodnie z ich definicjami oraz
| dodaj do tabeli "doctors"domyślnego administratora
end

```
1       def _setup_database_object(self) -> None:
2         try:
3             self.db = SqliteCipher(
4                 dataBasePath=os.getenv("DB_PATH"),
5                 checkSameThread=False,
6                 password=os.getenv("DB_PASSWORD"),
7             )
8         except Exception as ex:
9             logging.error(ex)
10
11        for table_name in ["diabetes", "doctors", "patients"]:
12            if not self.db.checkTableExist(table_name):
13                self._init_database()
14                break
15
16    def _init_database(self) -> None:
17        self.db.createTable(
18            "diabetes",
19            [
20                ["pregnancies", "INT"],
21                ["glucose", "REAL"],
22                ["blood_pressure", "REAL"],
23                ["skin_thickness", "REAL"],
24                ["insulin", "REAL"],
25                ["bmi", "REAL"],
26                ["diabetes_pedigree_function", "REAL"],
27                ["age", "INT"],
28                ["outcome", "INT"],
29            ],
30            True,
31            True,
32        )
33
34        self.db.createTable(
35            "doctors",
36            [
37                ["first_name", "TEXT"],
38                ["last_name", "TEXT"],
39                ["email", "TEXT"],
40                ["phone_number", "TEXT"],
41                ["hashed_password", "TEXT"],
42            ],
43            True,
```

```
            True,
        )

        self.db.insertIntoTable(
            "doctors",
            [
                "Senior",
                "Registrar",
                "senior_registrar@hospital.com",
                "+48-111-222-333",
                password_utils.get_hashed_password(os.getenv("
                    DEV_PASSWORD")),
            ],
            True,
        )

        self.db.createTable(
            "patients",
            [
                ["PESEL", "TEXT"],
                ["first_name", "TEXT"],
                ["last_name", "TEXT"],
                ["email", "TEXT"],
                ["phone_number", "TEXT"],
                ["historical_data", "BLOB"],
            ],
            True,
            True,
        )
```

Elementy pomocnicze

Pliki **db_models.py**, **jwt_utils.py** oraz **password_utlis.py** zawierają modele oraz funkcje pomocnicze dla obsługi bazy danych oraz endpointów.

Testy

test_train_model_endpoint

- **Cel:** Testowanie endpointu odpowiedzialnego za trenowanie modelu uczenia maszynowego.
- **Proces:** Wysyłanie żądania POST na endpoint `/train` z plikiem `diabetes.csv`, zawierającym dane do treningu modelu. Używa tokena typu Bearer do autoryzacji.
- **Sprawdzenie:** Status odpowiedzi równy 200, co oznacza pomyślne przetworzenie żądania.
- **Wynik:** Wyświetla `"train 200"` po pomyślnym ukończeniu.

get_login

- **Cel:** Uzyskanie tokenu dostępu do autoryzacji w innych testach.
- **Proces:** Wysyłanie żądania POST na endpoint `/login` z danymi użytkownika. Funkcja zwraca cały obiekt odpowiedzi.

test_predict_diabetes_endpoint

- **Cel:** Testowanie endpointu do przewidywania cukrzycy.
- **Proces:** Wysyłanie żądania POST na endpoint `/predict` z danymi JSON zawierającymi informacje o pacjencie i jego parametrach zdrowotnych.
- **Sprawdzenie:** Status odpowiedzi równy 200.
- **Wynik:** Wyświetla `"predict: 200"` po pomyślnym ukończeniu.

test_get_data_endpoint

- **Cel:** Testowanie endpointu odpowiedzialnego za pobieranie danych.
- **Proces:** Wysyłanie żądania GET na endpoint `/data`.
- **Sprawdzenie:** Status odpowiedzi równy 200.
- **Wynik:** Wyświetla `"data: 200"` po pomyślnym ukończeniu.

test_signup_endpoint

- **Cel:** Testowanie endpointu do rejestracji użytkowników.
- **Proces:** Wysyłanie żądania POST na endpoint `/signup` z danymi nowego użytkownika.
- **Sprawdzenie:** Status odpowiedzi równy 200.
- **Wynik:** Wyświetla `"signup"`.

Wykonanie Testów

- Skrypt uzyskuje token dostępu poprzez wywołanie `get_login()`.
- Następnie sekwencyjnie wykonuje `test_get_data_endpoint`, `test_predict_diabetes_endpoint` i `test_train_model_endpoint`, przekazując uzyskany token do autoryzacji.

Testy pozwalały na weryfikację czy zmiany wprowadzanie podczas rozwoju aplikacji były bespieczne z punktu widzenia api w sposób ciągły

Pełen kod aplikacji

```
1 Tutaj wklejamy pelen kod.
```