Ćwiczenie laboratoryjne

10. Projektowanie aplikacji mobilnych do monitorowania zdrowia: zbieranie danych zdrowotnych i analiza za pomocą modeli uczenia maszynowego.

1 Projektowanie aplikacji mobilnych do monitorowania zdrowia

1.1 Cel zajęć

Celem zajęć jest zapoznanie studentów z podstawami projektowania aplikacji monitorujących stan zdrowia pacjenta, obejmującymi:

- zbieranie danych zdrowotnych,
- wstępną analizę i wizualizację danych,
- wykorzystanie prostych modeli uczenia maszynowego,
- projektowanie interfejsu użytkownika aplikacji zdrowotnej.

Zajęcia mają charakter **prototypowy** i koncentrują się na aspektach analitycznych oraz decyzyjnych, a nie na pełnym procesie inżynierii aplikacji mobilnych.

1.2 Aplikacja mobilna wykonana w Pythonie i Jupyter Notebook

W kontekście dydaktycznym aplikacja monitorująca zdrowie może zostać zaprojektowana i przetestowana z użyciem:

- języka **Python**,
- środowiska **Jupyter Notebook**,
- prostych frameworków webowych działających lokalnie.

Jupyter Notebook nie jest narzędziem do tworzenia natywnych aplikacji mobilnych, ale doskonale sprawdza się jako:

- środowisko prototypowania aplikacji zdrowotnych,

- narzędzie do analizy danych i trenowania modeli ML,
- interaktywna „aplikacja analityczna” symulująca działanie systemu mobilnego.

Takie podejście pozwala skupić się na logice aplikacji i analizie danych, bez konieczności nauki zaawansowanych technologii mobilnych.

1.3 Proponowane najprostsze narzędzie do tworzenia aplikacji

Najprostszym i najbardziej polecanym narzędziem do realizacji tego ćwiczenia jest:

Streamlit (Python). Streamlit umożliwia tworzenie prostych aplikacji webowych (działających również na smartfonach w przeglądarce) bez znajomości technologii frontendowych.

Zalety Streamlit:

- pełne wsparcie dla Pythona i modeli ML,
- bardzo niski próg wejścia,
- możliwość uruchamiania aplikacji lokalnie,
- interfejs przyjazny urządzeniom mobilnym,
- integracja z Jupyter Notebook (logika może być rozwijana w notebooku).

Alternatywy (opcjonalnie):

- **Kivy** – aplikacje mobilne w Pythonie (bardziej zaawansowane),
- **Flutter / React Native** – poza zakresem tych zajęć.

1.4 Zakres funkcjonalny aplikacji zdrowotnej

Projektowana aplikacja powinna umożliwiać:

- wprowadzanie danych zdrowotnych (np. wiek, BMI, ciśnienie, tętno),
- zapisywanie danych w lokalnej strukturze (CSV),
- wizualizację parametrów zdrowotnych,
- predykcję stanu zdrowia z użyciem prostego modelu ML,
- prezentację wyniku w czytelnej formie dla użytkownika.

1.5 Przebieg ćwiczenia

Etap 1: Zbieranie danych zdrowotnych

- Zdefiniuj zestaw danych wejściowych aplikacji (np. ciśnienie, BMI, glukoza).
- Przygotuj formularz wprowadzania danych (Streamlit lub notebook).
- Zapisz dane do pliku CSV.

Etap 2: Analiza i wizualizacja danych

- Oblicz podstawowe statystyki opisowe.
- Wykonaj wykresy trendów parametrów zdrowotnych.
- Zidentyfikuj wartości nieprawidłowe lub graniczne.

Etap 3: Model uczenia maszynowego

- Zbuduj prosty model klasyfikacyjny (np. regresja logistyczna).
- Przewiduj ryzyko zdrowotne na podstawie danych użytkownika.
- Oceń poprawność predykcji.

Etap 4: Integracja modelu z aplikacją

- Połącz model ML z interfejsem aplikacji.
- Prezentuj wynik predykcji w czasie rzeczywistym.
- Zadbaj o czytelną interpretację wyniku.

Etap 5: Aspekty bezpieczeństwa i etyki

- Omów zasady ochrony danych zdrowotnych.
- Zastosuj minimalizację danych.
- Podkreśl rolę aplikacji jako narzędzia wspomagającego, a nie diagnostycznego.

1.6 Przykładowy kod aplikacji (Streamlit)

```
import streamlit as st

st.title("Aplikacja monitorowania zdrowia")

age = st.number_input("Wiek", 18, 100)
sbp = st.number_input("Ciśnienie skurczowe", 80, 220)
dbp = st.number_input("Ciśnienie rozkurczowe", 40, 130)

if st.button("Analizuj"):
    if sbp >= 140 or dbp >= 90:
        st.warning("Podwyższone ryzyko nadciśnienia")
    else:
        st.success("Parametry w normie")
```

2 Przykładowa aplikacja mobilna (prototyp) w Streamlit — instalacja, uruchomienie i opis kodu

2.1 Założenia i cel przykładu

W ramach ćwiczenia aplikacja mobilna jest rozumiana jako **responsywna aplikacja webowa**, która działa w przeglądarce smartfona (Android/iOS). Taki prototyp pozwala zrealizować zbieranie danych zdrowotnych, analizę oraz predykcję modelem ML bez budowania natywnej aplikacji.

Do implementacji wykorzystujemy **Python + Streamlit**, co umożliwia:

- szybkie tworzenie interfejsu (formularze, przyciski, wykresy),
- zapis historii pomiarów do pliku CSV,
- trenowanie i uruchamianie modelu uczenia maszynowego,
- testowanie aplikacji na telefonie (wejście przez przeglądarkę).

2.2 Instalacja środowiska i bibliotek

Zaleca się wykonywanie poleceń w jednym, spójnym środowisku Pythona (np. wirtualne środowisko `venv` lub środowisko Conda).

Instalacja pakietów

Zainstaluj wymagane biblioteki:

```
python -m pip install streamlit pandas scikit-learn joblib matplotlib
```

Jeżeli system Windows zgłasza błąd:

```
'streamlit' is not recognized as an internal or external command
```

należy uruchamiać Streamlit w sposób niezależny od ustawień PATH:

```
python -m streamlit run app.py
```

2.3 Uruchomienie aplikacji

Krok 1: zapis kodu do pliku

Zapisz kod aplikacji do pliku o nazwie `app.py` w katalogu roboczym.

Krok 2: uruchomienie

Uruchom aplikację:

```
python -m streamlit run app.py
```

Po uruchomieniu w terminalu pojawi się adres lokalny, np.:

```
Local URL: http://localhost:8501
Network URL: http://192.168.0.10:8501
```

Krok 3: uruchomienie na telefonie

Aby otworzyć aplikację na smartfonie:

- komputer i telefon muszą być w tej samej sieci Wi-Fi,
- w przeglądarce telefonu wpisz adres `Network URL` (np. `http://192.168.0.10:8501`).

2.4 Struktura plików projektu

Aplikacja tworzy i wykorzystuje dwa pliki:

- `health_measurements.csv` – historia pomiarów (dane wejściowe użytkownika),
- `risk_model.joblib` – zapisany model ML oraz metryki jakości.

2.5 Opis działania aplikacji (mapowanie na etapy ćwiczenia)

Kod aplikacji realizuje Etapy 1–4:

1. **Etap 1 (Zbieranie danych):** formularz danych pacjenta, zapis do CSV.
2. **Etap 2 (Analiza):** statystyki opisowe i wykres trendów pomiarów.
3. **Etap 3 (Model ML):** trenowanie regresji logistycznej na historii pomiarów.
4. **Etap 4 (Integracja):** predykcja ryzyka dla bieżącego pomiaru w interfejsie.

2.6 Opis kluczowych fragmentów kodu aplikacji

Inicjalizacja aplikacji

Fragment:

```
st.set_page_config(page_title="Monitor zdrowia + ML", layout="centered")
st.title(" Monitor zdrowia + analiza ML (demo)")
```

Znaczenie:

- konfiguruje wygląd strony i tytuł aplikacji,
- przygotowuje interfejs użytkownika.

Etap 1: formularz wejściowy i zapis do CSV

Formularz w Streamlit:

```
with st.form("health_form"):
    age = st.number_input("Wiek", 18, 110, 40)
    bmi = st.number_input("BMI", 10.0, 60.0, 24.0)
    ...
    submitted = st.form_submit_button("Zapisz pomiar")
```

Zapis pomiaru:

```
row = {"timestamp": ..., "age": ..., "bmi": ..., ...}
append_measurement(row)
```

Znaczenie:

- użytkownik wprowadza dane zdrowotne,
- dane są dopisywane do pliku `health_measurements.csv`,
- historia pomiarów pozwala na dalszą analizę i trenowanie modelu.

Etap 2: analiza i wizualizacja

Statystyki opisowe:

```
df[["age","bmi","glucose","systolic_bp","diastolic_bp"]].describe()
```

Wykres trendu:

```
plt.plot(df_plot["timestamp"], df_plot["systolic_bp"], label="systolic_bp")
plt.plot(df_plot["timestamp"], df_plot["diastolic_bp"], label="diastolic_bp")
```

Znaczenie:

- aplikacja prezentuje wartości w czasie (monitoring),
- umożliwia wykrycie trendów i wartości skrajnych.

Etap 3: trenowanie modelu ML

Aplikacja trenuje prosty klasyfikator:

- model: regresja logistyczna,
- cechy: `age, bmi, glucose, systolic_bp, diastolic_bp`,
- etykieta demonstracyjna: „podwyższone ryzyko” z progów SBP/DBP (wyłącznie dydaktycznie).

Pipeline (przetwarzanie + model):

```
clf = Pipeline(steps=[
  ("pre", preprocessor),
  ("model", LogisticRegression(max_iter=2000))
])
```

Znaczenie:

- dane są imputowane i skalowane,
- model uczy się zależności na podstawie historii pomiarów.

Etap 4: predykcja w interfejsie

Predykcja dla bieżącego wpisu:

```
proba = model.predict_proba(X_one)[0, 1]
pred = int(proba >= 0.5)
```

Znaczenie:

- aplikacja oblicza prawdopodobieństwo klasy „podwyższone ryzyko”,
- wynik jest prezentowany użytkownikowi jako komunikat.

2.7 Uwagi bezpieczeństwa i odpowiedzialności

Aplikacja stanowi **demonstrację edukacyjną** i nie jest narzędziem diagnostycznym. W praktycznych zastosowaniach medycznych należałoby zapewnić m.in.:

- ochronę danych (szyfrowanie, kontrola dostępu),
- walidację kliniczną modelu,
- zgodność regulacyjną (RODO, standardy jakości, audyty).

2.8 Rozszerzenia (opcjonalne)

- dodanie kont użytkowników i logowania (autoryzacja),
- eksport raportu PDF z ostatnich pomiarów,
- alerty o przekroczeniach progów,
- zastąpienie etykiety progowej rzeczywistą diagnozą (dane kliniczne).

3 Warianty zadań dla studentów (na bazie aplikacji `app.py`)

Każdy wariant zakłada pracę na dostarczonym kodzie `app.py` (Streamlit) i obejmuje pełny przebieg ćwiczenia (Etapy 1–4):

1. **Etap 1:** zbieranie danych (formularz + zapis do CSV),
2. **Etap 2:** analiza i wizualizacja danych,
3. **Etap 3:** budowa/ocena modelu ML,
4. **Etap 4:** integracja predykcji z interfejsem aplikacji.

Wariant 1: Historia pomiarów i filtrowanie (MVP)

- Etap 1: dodaj wybór użytkownika (`user_id`) i zapisuj pomiary osobno dla użytkowników.
- Etap 2: dodaj filtr (wybór użytkownika) oraz zakres dat do analizy.
- Etap 3: trenuj model osobno dla wybranego użytkownika (jeśli liczba rekordów $\geq$ 20).
- Etap 4: predykcja dotyczy aktualnie wybranego użytkownika i jego historii.

Wariant 2: Walidacja danych wejściowych i obsługa błędów

- Etap 1: dodaj walidację zakresów (np. SBP 70–260, DBP 40–150, BMI 10–60) oraz komunikaty o błędach.
- Etap 2: wykryj i oznacz wartości odstające (np. IQR) w tabeli i na wykresie.
- Etap 3: porównaj metryki modelu przed i po usunięciu wartości odstających.
- Etap 4: jeśli dane wejściowe są niewiarygodne, zablokuj predykcję i pokaż uzasadnienie.

Wariant 3: Trendy tygodniowe i średnie kroczące

- Etap 1: dodaj pole `note` (krótka notatka użytkownika) i zapisz do CSV.
- Etap 2: dodaj średnią kroczącą (np. 7 ostatnich pomiarów) na wykresie SBP/DBP.
- Etap 3: użyj cech pochodnych (np. średnia krocząca SBP, DBP) w modelu.
- Etap 4: pokaż predykcję oraz trend (czy ryzyko rośnie/maleje).

Wariant 4: Klasy ryzyka (3 poziomy) zamiast binarnego

- Etap 1: zapisuj dodatkowo `pulse` (tętno) i `steps` (kroki dzienne, opcjonalnie).
- Etap 2: wizualizuj rozkład klas ryzyka (low/medium/high) w oparciu o progi.
- Etap 3: zmień etykietę na 3 klasy i trenuj klasyfikator wieloklasowy (np. `LogisticRegression` multinomial).
- Etap 4: pokazuj użytkownikowi klasę i krótką interpretację zaleceń (edukacyjnie).

Wariant 5: Predykcja regresyjna (np. SBP jutro) – wersja uproszczona

- Etap 1: zbieraj dane w trybie „dziennym” (jedna próbka dziennie) i pilnuj spójnych timestampów.
- Etap 2: dodaj wykres różnic dzień-do-dnia dla SBP (delta).

- Etap 3: zamiast klasyfikacji wykonaj regresję (np. `Ridge`) przewidując SBP na podstawie cech.
- Etap 4: pokaż przewidywane SBP oraz błąd walidacyjny (MAE) w interfejsie.

Wariant 6: Porównanie dwóch modeli ML

- Etap 1: bez zmian lub dodaj dodatkową cechę (np. `sleep_hours`).
- Etap 2: dodaj macierz korelacji (dla cech numerycznych) i krótki komentarz.
- Etap 3: wytrenuj dwa modele (np. `LogisticRegression` i `RandomForestClassifier`) i porównaj AUC/Accuracy.
- Etap 4: w UI dodaj wybór aktywnego modelu do predykcji (przełącznik).

Wariant 7: Interpretacja predykcji (proste wyjaśnienie bez SHAP)

- Etap 1: dodaj checkbox „czy pomiar po wysiłku" i zapisz do CSV.
- Etap 2: pokaż histogramy cech oraz porównanie dla klas ryzyka.
- Etap 3: po treningu oblicz Permutation Importance dla cech.
- Etap 4: w UI pokaż 3 najważniejsze cechy i krótki komentarz do predykcji.

Wariant 8: Tryb anonimowy vs spersonalizowany

- Etap 1: dodaj wybór trybu: anonimowy (bez `user_id`) lub spersonalizowany (z `user_id`).
- Etap 2: porównaj statystyki populacyjne vs użytkownika.
- Etap 3: trenuj model globalny (wszyscy) oraz lokalny (użytkownik) i porównaj metryki.
- Etap 4: predykcja przełącza się zależnie od trybu, a UI informuje o różnicach.

Wariant 9: Alerty i progi bezpieczeństwa

- Etap 1: dodaj pole „objawy" (lista wyboru) i zapis do CSV.
- Etap 2: wykryj i oznacz pomiary alarmowe (np. SBP $\geq$ 180 lub DBP $\geq$ 110).
- Etap 3: dopasuj próg decyzji modelu (np. 0.7 zamiast 0.5) i oceń wpływ na czułość.
- Etap 4: jeśli alarm, pokaż czerwony alert i instrukcję postępowania (edukacyjnie).

Wariant 10: Eksport raportu z ostatnich pomiarów

- Etap 1: dodaj przycisk „pobierz historię” (eksport CSV filtrowany).
- Etap 2: generuj podsumowanie (min/mean/max) dla ostatnich 30 dni.
- Etap 3: trenuj model i zapisuj metryki w pliku obok modelu.
- Etap 4: w UI dodaj sekcję „Raport” (podsumowanie + wynik predykcji).

Wariant 11: Uczenie przyrostowe (symulacja) i wersjonowanie modelu

- Etap 1: po każdym nowym pomiarze dopisz licznik wersji danych.
- Etap 2: pokaż na wykresie momenty, w których trenowano model (markery).
- Etap 3: zapisuj modele z numerem wersji (np. `risk_model_v3.joblib`).
- Etap 4: w UI umożliw wybór wersji modelu do predykcji (lista).

Wariant 12: Jakość danych i imputacja braków

- Etap 1: pozwól użytkownikowi pominąć jedną cechę (np. BMI) i zapisuj brak jako NaN.
- Etap 2: raportuj procent braków i pokaż, jak imputacja wpływa na statystyki.
- Etap 3: porównaj dwie strategie imputacji (median vs mean) i ich wpływ na metryki.
- Etap 4: w predykcji pokaż użytkownikowi, które pola zostały imputowane.

Wariant 13: Uproszczona personalizacja progów

- Etap 1: dodaj możliwość ustawienia własnych progów (np. SBP alarmowe) w UI i zapisz do pliku konfiguracyjnego.
- Etap 2: wizualizuj przekroczenia względem progów użytkownika.
- Etap 3: trenuj model i porównaj działanie progów vs predykcji ML.
- Etap 4: pokaż różnicę: „progowo” vs „ML” dla bieżącego pomiaru.

Wariant 14: Analiza segmentów pacjentów (grupy)

- Etap 1: dodaj pole „płeć” i zapis do CSV (kategoria).
- Etap 2: porównaj statystyki i wykresy w segmentach (np. płeć lub grupy wieku).
- Etap 3: użyj kodowania kategorii i porównaj metryki modelu dla segmentów.
- Etap 4: w UI pokaż predykcję oraz informację, do jakiego segmentu należy użytkownik.

Wariant 15: Kompletny prototyp „health monitoring”

- Etap 1: dodaj minimum 2 nowe parametry (np. `pulse`, `sleep_hours`) i zapis.
- Etap 2: dashboard: wykresy, histogramy, alerty, korelacje.
- Etap 3: porównaj co najmniej 2 modele + metryki + progi decyzyjne.
- Etap 4: integracja: wybór modelu, predykcja, interpretacja (top cechy), raport.

Wymagania wspólne (dla każdego wariantu).

- Kod ma działać po uruchomieniu: `python -m streamlit run app.py`.
- Każdy wariant musi zapisywać dane do CSV oraz wykonać analizę (co najmniej 1 wykres).
- Model ML musi być trenowany i oceniony (co najmniej jedna metryka).
- Predykcja musi być zintegrowana z interfejsem aplikacji i opisana komunikatem dla użytkownika.

3.1 Podsumowanie

Zastosowanie Pythona, Jupyter Notebook oraz Streamlit pozwala w prosty sposób zaprojektować prototyp aplikacji mobilnej do monitorowania zdrowia. Takie podejście umożliwia skupienie się na analizie danych, modelach ML oraz interpretacji wyników, bez konieczności znajomości zaawansowanych technologii mobilnych.