Optymalizacja wyboru miejsca parkingowego dla studenta i prowadzącego

Tomasz Maj

1 lipca 2025

Streszczenie

Celem sprawozdania jest przedstawienie modelu matematycznego do optymalizacji wyboru miejsca parkingowego dla przyjeżdżającego studenta. Model uwzględnia dostępność miejsc parkingowych, czas dojazdu z domu do parkingu, czas dojścia z parkingu do budynku oraz ryzyko braku wolnego miejsca, które jest funkcją udziału wolnych miejsc w całkowitej pojemności parkingu.

1 Wprowadzenie

W problemie optymalizacji wyboru parkingu dla studenta istotne są następujące czynniki:

- dostępność miejsc parkingowych na pięciu różnych parkingach,
- czas dojazdu z domu do parkingu,
- czas dojścia z parkingu do budynku,
- czas dojścia z budynku do parkingu,
- ryzyko braku miejsca, które zależy od liczby wolnych miejsc na parkingu.

W niniejszym sprawozdaniu zaproponowany zostanie model matematyczny uwzględniający powyższe czynniki oraz sposób jego implementacji.

2 Pozyskiwanie danych

Dane o dostępności miejsc parkingowych pozyskiwane są przez skrypt w Pythonie wykorzystujący API systemu *iParking* Politechniki Wrocławskiej. Dane są przetwarzane i zapisywane do lokalnej bazy danych MS SQL.

Fragment kodu scrapującego dane:

```
payload_info  = {"o": "get_parks", "i": 2}
response_info = requests.post(url, headers=headers, data=json.
   dumps(payload_info))
parking_id = [i['id'] for i in response_info.json()['places']]
```

3 Model predykcyjny

Do prognozowania liczby dostępnych miejsc wykorzystano prostą sieć neuronową (MLP) zbudowaną w PyTorch. Model uczony jest na podstawie danych historycznych. Dane wejściowe zawierają: identyfikator parkingu, dzień, godzinę, minutę, dzień tygodnia, miesiąc i rok.

Architektura modelu:

```
class RegressionModel(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, 32)
        self.fc2 = nn.Linear(32, 8)
        self.fc3 = nn.Linear(8, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x); x = self.relu(x)
        x = self.fc2(x); x = self.relu(x)
        x = self.fc3(x)
        return x
```

Model osiągał sensowne wartości błędu średniokwadratowego (MSE), umożliwiając jego zastosowanie w etapie decyzyjnym.

4 Model matematyczny

Niech $i = 1, \ldots, 5$ oznacza numer parkingu.

4.1 Zmienne i parametry

- $x_i \in \{0, 1\}$ — zmienna decyzyjna, gdzie $x_i = 1$ oznacza wybór parkingu i, a $x_i = 0$ jego odrzucenie.
- t_i^d — czas dojazdu z domu do parkingu i (w minutach).
- t_i^w — czas dojścia z parkingu i do budynku (w minutach).
- t_i^p — czas dojścia z budynku do parkingu i (w minutach).
- a_i — liczba wolnych miejsc na parkingu i.
- c_i — całkowita liczba miejsc na parkingu i.
- α — współczynnik ryzyka braku miejsca.
- β — współczynnik wpływu dostępności miejsc na ryzyko.

4.2 Definicja ryzyka

Ryzyko braku miejsca na parkingu i definiujemy jako funkcję udziału wolnych miejsc w całkowitej pojemności parkingu. Im mniej miejsc dostępnych, tym większe ryzyko:

$$r_i = \frac{t_i^d}{\left(\frac{a_i}{c_i} \cdot \beta\right) + 1} \tag{1}$$

4.3 Funkcja celu

Celem jest minimalizacja sumarycznego kosztu, który obejmuje łączny czas dojazdu i dojścia oraz ryzyko braku miejsca:

$$\min \sum_{i=1}^{5} x_i \cdot \left(t_i^d + t_i^w + t_i^p + \alpha \cdot \frac{t_i^d}{\left(\frac{a_i}{c_i} \cdot \beta\right) + 1} \right) \tag{2}$$

4.4 Ograniczenia

$$\sum_{i=1}^{5} x_i = 1 \qquad \text{(wybieramy dokładnie jeden parking)} \tag{3}$$

$$x_i = 0 \quad \text{jeśli } a_i = 0 \qquad \text{(brak dostępnych miejsc)} \tag{4}$$

$$x_i \in \{0, 1\} \qquad \text{(zmienne binarne)} \tag{5}$$

5 Implementacja heurystyczna w Pythonie

Przykładowy kod wyboru parkingu:

```
def get_park_optimization(first_building, last_building,
    df_results):

    equation_dict = {}
    parks = df_park_info['park_id'].values.tolist()
    car_park_times = get_time_car_park()
    park_slots = get_park_prediction(df_results)


    for park in parks:
        totals = df_park_info[df_park_info['park_id'] == park
            ]['park_total'].values
        alfa = 10
        beta = 30
        park_building_times = get_time_park_building(park,
            first_building)
        building_park_times = get_time_park_building(park,
            last_building)
```

```
15
16          car_park_time = car_park_times[f"car_{park}"]
17          park_slots_available = park_slots[f"{park}"]
18          r = car_park_time/((park_slots_available/totals[0])*
                beta + 1)# risk score
19          if park_slots_available > 0:
20              equation = park_building_times +
                    building_park_times + car_park_time + (alfa * r
                    )
21              equation_dict[park] = equation
22
23      return equation_dict
```

6 Podsumowanie

W sprawozdaniu przedstawiono model matematyczny optymalizacji wyboru miejsca parkingowego z uwzględnieniem czasu dojazdu, dojścia oraz ryzyka braku wolnych miejsc. Zaprezentowano funkcję celu oraz możliwą implementację heurystyczną. Takie podejście może być wykorzystane w aplikacjach wspomagających codzienne decyzje logistyczne.

6.1 Plany rozwoju

W przyszłości planowane są następujące rozszerzenia systemu:

- Stworzenie aplikacji umożliwiającej użytkownikom bezpośrednie korzystanie z możliwości modelu optymalizacyjnego.
- Obsługa wielu użytkowników jednocześnie, co będzie wymagało implementacji mechanizmu szeregowania i przydzielania parkingów w czasie rzeczywistym.
- Udoskonalenie sposobu pozyskiwania czasu dojazdu — zamiast wykorzystywania stałej wartości roboczej, planowane jest wykorzystanie danych GPS w czasie rzeczywistym (np. poprzez integrację z Google Maps API).
- Zastosowanie bardziej zaawansowanego algorytmu optymalizacyjnego, który lepiej uwzględnia ryzyko braku miejsca, np. poprzez użycie funkcji ryzyka opartej na rozkładzie wykładniczym.