UNIWERSYTET RZESZOWSKI
Wydział Nauk Ścisłych i Technicznych

Piotr Smoła
nr albumu: 125162
Kierunek: Informatyka

System rekomendacji produktów wykorzystujący filtrację opartą na treści, logikę rozmytą i modele probabilistyczne

Praca inżynierska

Praca wykonana pod kierunkiem
dr inż. Piotra Grochowalskiego

Rzeszów, 2026

Spis treści

Wstęp

Nowoczesne platformy e-commerce oferują tysiące lub dziesiątki tysięcy produktów, co stanowi istotne wyzwanie zarówno dla klientów, jak i właścicieli sklepów internetowych. Użytkownik poszukujący produktu staje przed wyborem setek opcji, co często prowadzi do rezygnacji z zakupu. Bez wsparcia inteligentnych systemów rekomendacyjnych proces zakupowy staje się czasochłonny i frustrujący. Z perspektywy biznesowej oznacza to utratę potencjalnych klientów oraz sytuacje, w których nabywcy nie odkrywają produktów optymalnie dopasowanych do ich potrzeb.

Systemy rekomendacyjne stanowią rozwiązanie tego problemu poprzez automatyczną analizę preferencji użytkowników w celu proponowania produktów dopasowanych do indywidualnych potrzeb, zwiększając jednocześnie konwersję i wartość sprzedaży [6].

System został zrealizowany jako część projektu e-commerce powstałego we współpracy dwuosobowej. Niniejsza praca koncentruje się na implementacji trzech metod rekomendacyjnych: filtracji opartej na treści (Content-Based Filtering), logiki rozmytej (Fuzzy Logic) oraz modeli probabilistycznych (łańcuch Markowa i naiwny klasyfikator Bayesa).

Motywacja i kontekst problemu

Problem rekomendacji w platformach e-commerce jest złożony i wieloaspektowy. Preferencje użytkowników są subiektywne i trudne do modelowania matematycznego. Dane o użytkownikach są często niekompletne lub niedostępne, zwłaszcza dla nowych użytkowników i produktów - zjawisko znane jako problem zimnego startu (cold start problem). Katalogi produktów dynamicznie się zmieniają, co wymaga ciągłej aktualizacji modeli rekomendacyjnych. Dodatkowo, użytkownicy oczekują nie tylko trafnych rekomendacji, ale również ich wyjaśnienia - dlaczego dany produkt został zaproponowany.

Istniejące rozwiązania komercyjne (Amazon Personalize, Google Recommendations AI) oferują zaawansowane mechanizmy rekomendacji, jednak działają jako czarne skrzynki (black-box), nie pozwalając na kontrolę nad algorytmami ani ich dostosowanie do specyficznych wymagań biznesowych. Biblioteki open-source (Apache Mahout, Surprise) rozwiązują problem kosztów, ale nie oferują implementacji logiki rozmytej ani zaawansowanych modeli probabilistycznych w jednym spójnym systemie.

Rzeczona praca odpowiada na powyższe wyzwania poprzez zaprojektowanie i implementację modułowego systemu rekomendacyjnego łączącego trzy komplementarne podejścia: filtrację opartą na treści (Content-Based Filtering, CBF), logikę

rozmytą (Fuzzy Logic) oraz modele probabilistyczne (łańcuchy Markowa i naiwny klasyfikator Bayesa). Każda z tych metod rozwiązuje inne aspekty problemu rekomendacji i oferuje unikalne możliwości personalizacji. System został zaimplementowany od podstaw, bez wykorzystania zewnętrznych bibliotek uczenia maszynowego, co zapewnia pełną kontrolę nad algorytmami oraz możliwość ich dostosowania do specyficznych wymagań platformy e-commerce.

Cel i zakres pracy

Głównym celem pracy jest zaprojektowanie i implementacja kompleksowego systemu rekomendacji produktów dla platformy e-commerce, który łączy sześć zaawansowanych i komplementarnych metod uczenia maszynowego. System został zbudowany od podstaw, z ograniczeniem wykorzystania gotowych bibliotek uczenia maszynowego, co zapewniło pełną kontrolę nad algorytmami i możliwość ich dostosowania do specyficznych wymagań biznesowych.

W ramach realizacji celu głównego zdefiniowano następujące cele szczegółowe:

- Zaprojektowanie architektury modułowego systemu rekomendacyjnego z architekturą trójwarstwową (Django + React + PostgreSQL)
- Implementacja algorytmu filtracji opartej na treści (Content-Based Filtering) z wykorzystaniem ważonych wektorów cech (kategorie 40%, tagi 30%, cena 20%, słowa kluczowe 10%) i miary podobieństwa kosinusowego
- Opracowanie systemu wnioskowania rozmytego typu Mamdani z regułami IF-THEN i funkcjami przynależności (trójkątne i trapezoidalne) dla trzech wymiarów: wrażliwość cenowa, preferencje kategorialne, preferencja jakości
- Zbudowanie modeli probabilistycznych: łańcucha Markowa pierwszego rzędu dla predykcji sekwencji zakupowych oraz naiwnego klasyfikatora Bayesa dla predykcji churnu i prawdopodobieństwa zakupu
- Optymalizacja wydajności systemu dla wdrożenia produkcyjnego (cache, bulk operations, indeksy bazodanowe, Docker)
- Przeprowadzenie ewaluacji jakości rekomendacji na rzeczywistych danych (500 produktów, 20 użytkowników, 265 zamówień)

Zakres pracy obejmuje trzy główne obszary: podstawy teoretyczne metod rekomendacyjnych, projekt i implementację systemu oraz ewaluację jego działania. System został zintegrowany z aplikacją webową e-commerce (Django 5.1.4 + React 18 + PostgreSQL 14), umożliwiającą weryfikację zaimplementowanych algorytmów w warunkach zbliżonych do rzeczywistych.

Niniejsza praca koncentruje się szczegółowo na aspektach algorytmicznych i jakościowych trzech metod, z uwzględnieniem problemu zimnego startu, interpretowalności wyników oraz personalizacji doświadczenia użytkownika:

- **Content-Based Filtering** - rozwiązuje problem zimnego startu dla nowych produktów poprzez analizę cech produktowych
- **Fuzzy Logic** - oferuje najwyższą interpretowalność decyzji algorytmu poprzez reguły IF-THEN
- **Modele probabilistyczne** - przewidują sekwencje zakupowe i ryzyko churnu na podstawie historii transakcji

Rozdział 1

Teoretyczne podstawy metod rekomendacyjnych

System rekomendacji produktów został oparty na trzech zaawansowanych metodach uczenia maszynowego. Niniejszy rozdział prezentuje podstawy teoretyczne: filtracji opartej na treści (Content-Based Filtering), logiki rozmytej (Fuzzy Logic) oraz modeli probabilistycznych (Markov Chain, Naive Bayes).

1.1 Historia i ewolucja systemów rekomendacyjnych

Systemy rekomendacyjne powstały jako rozwiązanie problemu wyboru spośród dziesiątek tysięcy produktów dostępnych w sklepach internetowych. Rozwój tej dziedziny datuje się na lata 90. XX wieku, kiedy pojawiły się pierwsze platformy handlu elektronicznego. Wczesne komercyjne zastosowania systemów rekomendacji, w szczególności w Amazon.com, zostały udokumentowane w literaturze naukowej, stanowiąc punkt odniesienia dla kolejnych implementacji. Kluczowym przełomem była publikacja wprowadzająca metodę Item-Based Collaborative Filtering wykorzystującą metrykę Adjusted Cosine Similarity, która stała się standardem w przemyśle e-commerce.

Istotnym katalizatorem rozwoju zaawansowanych technik rekomendacji były konkursy badawcze, szczególnie Netflix Prize organizowany w latach 2006-2009. Konkurs ten, z pulą nagród wynoszącą milion dolarów, przyciągnął uwagę środowiska akademickiego oraz przemysłowego, znacząco przyspieszając rozwój algorytmów faktoryzacji macierzy, zespołów modeli oraz metod deep learning. Współcześnie systemy rekomendacyjne stanowią fundament funkcjonowania wiodących platform e-commerce (handlu elektronicznego), serwisów VOD (Video on Demand - wideo na żądanie), platform muzycznych oraz mediów społecznościowych.

Ewolucja systemów rekomendacyjnych przebiegała od prostych metod statystycznych (najbardziej popularne produkty, najlepiej ocenione) przez collaborative filtering (analiza wzorców zakupowych użytkowników), content-based filtering (analiza cech produktów), aż po zaawansowane metody hybrydowe łączące multiple podejścia. Obecne trendy obejmują wykorzystanie deep learning (sieci neuronowe), contextual bandits (algorytmy balansujące eksplorację i eksploatację), explainable AI (interpretowalność decyzji algorytmicznych) oraz personalizację w czasie rzeczywistym.

1.2 Content-Based Filtering – podstawy teoretyczne

Content-Based Filtering (CBF, filtracja oparta na treści) jest jedną z fundamentalnych metod systemów rekomendacyjnych. W przeciwieństwie do Collaborative Filtering, CBF analizuje cechy samych produktów, a nie wzorce zachowań użytkowników. Metoda została szczegółowo opisana w literaturze [1].

Zasada działania: System buduje profil cech każdego produktu (wektor cech) i oblicza podobieństwo między produktami na podstawie ich cech. Użytkownikowi rekomendowane są produkty podobne do tych, które wcześniej przeglądał lub kupił.

Reprezentacja produktu jako wektora cech

Każdy produkt p jest reprezentowany jako wektor w wielowymiarowej przestrzeni cech:

$$\vec{p} = (f_1, f_2, ..., f_n) \tag{1}$$

gdzie f_i to waga cechy i (np. należenie do kategorii, posiadanie tagu, przedział cenowy). W ogólnym przypadku stosuje się wagi różnicujące znaczenie poszczególnych cech:

$$\vec{p} = \sum_i w_i \cdot f_i(p) \tag{2}$$

gdzie w_i to waga cechy i, a $f_i(p)$ to wartość cechy dla produktu p. Funkcja indykatorowa $\mathbf{1}_{feature}(p)$ przyjmuje wartość 1 jeśli produkt posiada daną cechę, 0 w przeciwnym razie.

Zalety CBF:

- Brak problemu zimnego startu dla nowych produktów – wystarczy opis i cechy
- Przejrzystość rekomendacji – można wyjaśnić dlaczego produkt został polecony ("podobna kategoria", "podobne tagi")
- Niezależność od innych użytkowników – działa nawet dla pierwszego klienta w systemie
- Szybka aktualizacja – dodanie nowego produktu nie wymaga przeliczenia całej macierzy

Wady CBF:

- Problem "filter bubble" – rekomenduje tylko podobne produkty, użytkownik nie odkrywa nowych kategorii
- Wymaga dobrze opisanych cech produktów – jakość rekomendacji zależy od jakości metadanych
- Nie odkrywa nieoczywistych powiązań między produktami (np. "użytkownicy kupujący kawę często kupują cukier")
- Ograniczenie do podobieństwa cech – nie uwzględnia kontekstu użytkownika

Podobieństwo kosinusowe (Cosine Similarity) jest standardową metryką w CBF [9]. Dla dwóch wektorów $\vec{A}$ i $\vec{B}$:

$$\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| \times ||\vec{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3}$$

gdzie $\vec{A}$ i $\vec{B}$ to wektory cech dwóch produktów. Wynik mieści się w przedziale $[0, 1]$ dla nieujemnych wektorów (w kontekście TF-IDF i wag binarnych).

Interpretacja podobieństwa kosinusowego:

- $cos(\theta) = 1$ – wektory identyczne (produkty mają te same cechy)
- $cos(\theta) = 0$ – wektory ortogonalne (brak wspólnych cech)
- $cos(\theta) \in (0, 1)$ – częściowe podobieństwo

TF-IDF (Term Frequency - Inverse Document Frequency)

W kontekście ekstrakcji słów kluczowych z opisów tekstowych stosuje się miarę TF-IDF (Term Frequency - Inverse Document Frequency) [9]:

$$TF(t, d) = \frac{count(t, d)}{|d|} \tag{4}$$

gdzie $count(t, d)$ to liczba wystąpień terminu t w dokumencie d, a $|d|$ to długość dokumentu (liczba słów).

Pełna wersja TF-IDF uwzględnia rzadkość terminu w całym korpusie:

$$TF\text{-}IDF(t, d, D) = TF(t, d) \times IDF(t, D) \tag{5}$$

gdzie $IDF(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$ to odwrócona częstość dokumentowa, penalizująca terminy występujące w wielu dokumentach.

1.3 Logika rozmyta – podstawy teoretyczne

Logika rozmyta (Fuzzy Logic) została wprowadzona przez Lotfi Zadeha w przełomowej pracy [2]. Rozszerza klasyczną logikę dwuwartościową (prawda/fałsz) o stopnie przynależności w przedziale $[0, 1]$.

Motywacja: Klasyczna logika wymaga precyzyjnych granic. Pytanie "Czy produkt za 450 PLN jest tani?" nie ma jednoznacznej odpowiedzi – zależy od kontekstu, kategorii produktu i preferencji użytkownika. Logika rozmyta pozwala odpowiedzieć: "Produkt jest tani ze stopniem 0.3 i średnio drogi ze stopniem 0.7".

Zbiory rozmyte (Fuzzy Sets): W klasycznej teorii zbiorów element należy lub nie należy do zbioru. W zbiorach rozmytych element ma stopień przynależności $\mu(x) \in [0, 1]$. Formalnie, zbiór rozmyty A na uniwersum X jest zdefiniowany przez funkcję przynależności:

$$\mu_A : X \to [0, 1] \tag{6}$$

gdzie $\mu_A(x)$ oznacza stopień przynależności elementu x do zbioru A.

Przykład: Dla zmiennej "cena" możemy zdefiniować trzy zbiory rozmyte:

- **cheap**: ceny niskie (pełna przynależność dla cen < 100 PLN)
- **medium**: ceny średnie (pełna przynależność dla cen 500-1200 PLN)
- **expensive**: ceny wysokie (pełna przynależność dla cen > 2000 PLN)

Produkt za 350 PLN może mieć: $\mu_{cheap}(350) = 0.3$, $\mu_{medium}(350) = 0.5$, $\mu_{expensive}(350) = 0.0$.

Funkcje przynależności (Membership Functions) definiują stopień przynależności elementu do zbioru rozmytego. Najczęściej stosowane typy:

Funkcja trójkątna (Triangular MF):

$$\mu_{triangle}(x; a, b, c) = \max\left(0, \min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right)\right) \tag{7}$$

gdzie a to dolna granica, b to punkt maksymalny ($\mu = 1$), c to górna granica.

Funkcja trapezoidalna (Trapezoidal MF):

$$\mu_{trapezoid}(x; a, b, c, d) = \max\left(0, \min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right)\right) \tag{8}$$

gdzie przedział $[b, c]$ ma pełną przynależność ($\mu = 1$), a $[a, b)$ i $(c, d]$ to obszary przejściowe.

Funkcja gaussowska (Gaussian MF):

$$\mu_{gaussian}(x; c, \sigma) = e^{-\frac{(x-c)^2}{2\sigma^2}} \tag{9}$$

gdzie c to środek (mean), a σ to odchylenie standardowe kontrolujące szerokość.

Operacje na zbiorach rozmytych

Uzupełnienie (Negacja):

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{10}$$

Przecięcie (AND) – T-norma:

$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) \tag{11}$$

Najczęściej używane T-normy:

- Minimum (Gödel): $T_{min}(a, b) = \min(a, b)$
- Iloczyn algebraiczny: $T_{prod}(a, b) = a \cdot b$
- Łukasiewicz: $T_L(a, b) = \max(0, a + b - 1)$

Suma (OR) – T-conorma (S-norma):

$$\mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)) \tag{12}$$

Najczęściej używane T-conormy:

- Maksimum: $S_{max}(a, b) = \max(a, b)$
- Suma algebraiczna: $S_{sum}(a, b) = a + b - a \cdot b$
- Łukasiewicz: $S_L(a, b) = \min(1, a + b)$

System wnioskowania Mamdani [3] jest najbardziej rozpowszechnioną metodą wnioskowania rozmytego. Składa się z czterech etapów:

1. **Fuzzyfikacja** – przekształcenie wartości wejściowych na stopnie przynależności do zbiorów rozmytych. Przykład: cena 450 PLN $\rightarrow \mu_{cheap} = 0.1$, $\mu_{medium} = 0.6$, $\mu_{expensive} = 0.0$.
2. **Ewaluacja reguł** – obliczenie aktywacji reguł IF-THEN za pomocą T-norm. Dla reguły "IF price IS cheap AND quality IS high THEN recommendation IS strong":

$$\alpha = T(\mu_{cheap}(price), \mu_{high}(quality)) = \min(\mu_{cheap}, \mu_{high}) \tag{13}$$

3. **Agregacja** – połączenie wyników wszystkich reguł za pomocą T-conormy. Jeśli wiele reguł prowadzi do tego samego konsekwentu:

$$\mu_{output} = S(\alpha_1, \alpha_2, ..., \alpha_n) = \max(\alpha_1, \alpha_2, ..., \alpha_n) \tag{14}$$

4. **Defuzzyfikacja** – przekształcenie wyniku rozmytego na wartość liczbową.

   Metody defuzzyfikacji:
   Centroid (środek ciężkości):

$$y^* = \frac{\int y \cdot \mu(y) dy}{\int \mu(y) dy} \tag{15}$$

Średnia ważona (Weighted Average) – uproszczona metoda używana w implementacji:

$$y^* = \frac{\sum_{i=1}^{n} \alpha_i \cdot w_i}{\sum_{i=1}^{n} w_i} \tag{16}$$

gdzie α_i to aktywacja reguły i, a w_i to waga reguły.
Mean of Maximum (MoM):

$$y^* = \frac{1}{|M|} \sum_{y \in M} y, \quad M = \{y : \mu(y) = \max_z \mu(z)\} \tag{17}$$

Reguły rozmyte IF-THEN
Reguła rozmyta ma postać [10]:

$$\text{IF } x_1 \text{ IS } A_1 \text{ AND } x_2 \text{ IS } A_2 \text{ THEN } y \text{ IS } B \tag{18}$$

gdzie A_1, A_2, B to zbiory rozmyte definiujące warunki i konsekwencje reguły. Operator AND realizowany jest przez T-normę, najczęściej minimum lub iloczyn algebraiczny.

1.4 Modele probabilistyczne – podstawy teoretyczne

Łańcuchy Markowa (Markov Chains) zostały wprowadzone przez Andrieja Markowa w 1906 roku. Są procesami stochastycznymi spełniającymi własność Markowa – przyszły stan zależy tylko od stanu obecnego, nie od historii [4].

Definicja formalna: Łańcuch Markowa to ciąg zmiennych losowych $X_0, X_1, X_2, ...$ przyjmujących wartości ze zbioru stanów $S = \{s_1, s_2, ..., s_n\}$, spełniający własność Markowa:

$$P(X_{t+1} = s_j | X_t = s_i, X_{t-1} = s_{i-1}, ..., X_0 = s_0) = P(X_{t+1} = s_j | X_t = s_i) \tag{19}$$

Oznacza to, że prawdopodobieństwo przejścia do stanu s_j zależy tylko od obecnego stanu s_i, nie od tego jak do niego dotarliśmy.

Macierz przejść (Transition Matrix) P zawiera prawdopodobieństwa przejść między stanami:

$$P_{ij} = P(X_{t+1} = s_j | X_t = s_i) \tag{20}$$

Macierz P spełnia warunki:

- $P_{ij} \geq 0$ dla wszystkich i, j
- $\sum_j P_{ij} = 1$ dla wszystkich i (wiersze sumują się do 1)

Estymacja prawdopodobieństw przejść z danych:

$$\hat{P}_{ij} = \frac{count(s_i \to s_j)}{\sum_k count(s_i \to s_k)} \tag{21}$$

gdzie $count(s_i \to s_j)$ to liczba obserwowanych przejść ze stanu s_i do stanu s_j.

Rozkład stacjonarny (Stationary Distribution) π spełnia:

$$\pi = \pi P, \quad \sum_i \pi_i = 1 \tag{22}$$

Jest to rozkład prawdopodobieństwa, który pozostaje niezmieniony po przejściu – reprezentuje długoterminowe prawdopodobieństwa przebywania w każdym stanie. Rozkład stacjonarny jest istotny w analizie długoterminowego zachowania systemu.

Naiwny klasyfikator Bayesa (Naive Bayes, NB) opiera się na twierdzeniu Bayesa z założeniem niezależności cech [5].

Twierdzenie Bayesa:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \tag{23}$$

gdzie:

- $P(C|X)$ – prawdopodobieństwo a posteriori klasy C przy cechach X
- $P(C)$ – prawdopodobieństwo a priori klasy C
- $P(X|C)$ – wiarygodność (likelihood) – prawdopodobieństwo obserwacji cech X w klasie C
- $P(X)$ – prawdopodobieństwo marginalne cech (stałe dla wszystkich klas)

Założenie naiwne (Naive assumption) – niezależność warunkowa cech:

$$P(X|C) = P(x_1, x_2, ..., x_n|C) = \prod_{i=1}^{n} P(x_i|C) \tag{24}$$

Założenie to jest "naiwne" bo w rzeczywistości cechy są często skorelowane. Jednak Naive Bayes działa zaskakująco dobrze w praktyce.

Klasyfikacja:

$$\hat{C} = \arg\max_{C} P(C) \prod_{i=1}^{n} P(x_i|C) \tag{25}$$

Ponieważ $P(X)$ jest stałe dla wszystkich klas, można je pominąć przy porównywaniu.

Problem zerowych prawdopodobieństw: Jeśli cecha x_i nie wystąpiła w klasie C w danych treningowych, to $P(x_i|C) = 0$, co zeruje całe prawdopodobieństwo.

Wygładzanie Laplace'a (Laplace Smoothing / Add-one Smoothing) rozwiązuje ten problem:

$$P(x_i = v|C) = \frac{count(x_i = v, C) + 1}{count(C) + |V_i|} \tag{26}$$

gdzie $|V_i|$ to liczba unikalnych wartości cechy x_i. Dodanie 1 do licznika i $|V|$ do mianownika zapewnia, że żadne prawdopodobieństwo nie będzie zerowe.

Logarytm dla stabilności numerycznej: Iloczyn wielu małych prawdopodobieństw prowadzi do underflow. Rozwiązanie – praca w przestrzeni logarytmów:

$$\log P(C|X) = \log P(C) + \sum_{i=1}^{n} \log P(x_i|C) + const \tag{27}$$

Warianty Naive Bayes:

- **Multinomial NB** – dla danych zliczeniowych (np. częstość słów)
- **Bernoulli NB** – dla cech binarnych (obecność/brak)
- **Gaussian NB** – dla cech ciągłych (zakłada rozkład normalny)

Naive Bayes jest szeroko stosowany w klasyfikacji tekstu, filtracji spamu oraz systemach rekomendacyjnych ze względu na prostotę implementacji i niskie wymagania obliczeniowe.

1.5 Metryki oceny systemów rekomendacyjnych

Ewaluacja systemów rekomendacyjnych wymaga odpowiednich metryk jakości. Najpopularniejsze:

Precision@K – jaka część top K rekomendacji była faktycznie kupiona/polubiona:

$$Precision@K = \frac{|Recommended@K \cap Relevant|}{K} \tag{28}$$

Recall@K – jaka część produktów istotnych dla użytkownika została trafiona:

$$Recall@K = \frac{|Recommended@K \cap Relevant|}{|Relevant|} \tag{29}$$

F1-Score – harmoniczna średnia Precision i Recall:

$$F1@K = 2 \cdot \frac{Precision@K \cdot Recall@K}{Precision@K + Recall@K} \tag{30}$$

Mean Reciprocal Rank (MRR) – pozycja pierwszego trafienia:

$$MRR = \frac{1}{|U|} \sum_{u \in U} \frac{1}{rank_u} \tag{31}$$

gdzie $rank_u$ to pozycja pierwszego istotnego produktu w rankingu dla użytkownika u.

Coverage – procent produktów, które system jest w stanie rekomendować:

$$Coverage = \frac{|\text{products with recommendations}|}{|\text{all products}|} \tag{32}$$

Rozdział 2

Weryfikacja i analiza rozwiązań alternatywnych

W celu uzasadnienia sensowności tworzenia dedykowanego systemu rekomendacji przeprowadzono analizę trzech reprezentatywnych rozwiązań rynkowych. Celem weryfikacji było zidentyfikowanie ograniczeń istniejących narzędzi oraz określenie wymagań dla planowanej aplikacji e-commerce.

2.1 Amazon Personalize

Amazon Personalize to zarządzana usługa AWS oferująca systemy rekomendacji oparte na algorytmach stosowanych w Amazon.com. System wykorzystuje deep learning (głębokie uczenie - wielowarstwowe sieci neuronowe) oraz collaborative filtering, oferując trzy typy rekomendacji: User Personalization (personalizacja użytkownika), Similar Items (podobne produkty) oraz Personalized Ranking (spersonalizowane rankowanie).

Kluczowe ograniczenia w kontekście planowanego rozwiązania:

- **Wysokie koszty operacyjne** - rozwiązania chmurowe wiążą się z regularnymi opłatami licencyjnymi, które mogą być znaczące dla małych i średnich platform e-commerce. Dla porównania, własna implementacja eliminuje te koszty przy zachowaniu kontroli nad funkcjonalnościami,
- **Brak natywnej analizy sentymentu, logiki rozmytej oraz modeli probabilistycznych** - Amazon Personalize koncentruje się wyłącznie na collaborative filtering i nie oferuje analizy opinii produktów, systemów rozmytych ani łańcuchów Markova. Wieloźródłowa agregacja sentymentu, logika rozmyta Mamdani oraz modele probabilistyczne wymagają integracji z dodatkowymi usługami AWS lub samodzielnej implementacji,
- **Vendor lock-in** (uzależnienie od dostawcy) - głęboka integracja z ekosystemem AWS (S3, Lambda, EventBridge) oznacza, że migracja do innej platformy wymaga przepisania całej architektury systemu,
- **Brak kontroli nad algorytmami** - system działa jako „czarna skrzynka" (black box), uniemożliwiając dostosowanie logiki rekomendacji. Przykład: niemożliwe jest zaimplementowanie ważonych wektorów cech dla Content-Based Filtering, funkcji przynależności dla logiki rozmytej czy zaawansowanych metryk podobieństwa,

- **Wymóg dużych zbiorów danych** - według dokumentacji AWS, system wymaga minimum 25000 interakcji dla zapewnienia wysokiej jakości rekomendacji. Dla nowych platform (cold start - problem zimnego startu) jakość jest ograniczona w początkowym okresie działania.

2.2 Google Recommendations AI (Vertex AI)

Google Recommendations AI to platforma GCP wykorzystująca deep learning oraz multi-armed contextual bandits (wieloramienne bandyty kontekstowe - algorytmy balansujące eksplorację nowych opcji z wykorzystaniem sprawdzonych rozwiązań). System oferuje zaawansowane rekomendacje dla e-commerce, VOD (Video on Demand - wideo na żądanie) oraz platform newsowych, z automatycznym wykrywaniem trendów i sezonowości.

Kluczowe ograniczenia w kontekście planowanego rozwiązania:

- **Bardzo wysokie koszty operacyjne** - koszty usług GCP są często wyższe niż konkurencyjnych rozwiązań chmurowych, co czyni je nieopłacalnymi dla małych i średnich platform e-commerce. Model cenowy oparty na liczbie predykcji generuje rosnące koszty wraz ze skalą ruchu użytkowników,
- **Brak wieloźródłowej agregacji sentymentu, logiki rozmytej i modeli probabilistycznych** - Google Recommendations AI oferuje funkcję "Frequently Bought Together" (często kupowane razem, podobną do algorytmu Apriori), ale nie wspiera agregacji sentymentu z wielu źródeł tekstowych, systemów rozmytych z funkcjami przynależności ani łańcuchów Markova jak w planowanym systemie,
- **Wymóg bardzo dużych zbiorów danych** - rozwiązanie zaprojektowane dla platform o skali YouTube, co czyni je nadmiarowo złożonym dla małych sklepów internetowych,
- **Brak interpretowalności** - głęboka „black box", gdzie nawet administratorzy z dostępem do Vertex AI nie mogą zobaczyć wag embeddings (reprezentacji wektorowych) ani logiki sieci neuronowej, co uniemożliwia debugowanie i optymalizację. W przeciwieństwie do tego, logika rozmyta oferuje pełną przejrzystość reguł IF-THEN.

2.3 Apache Mahout

Apache Mahout to open-source framework (otwartoźródłowy framework) implementujący klasyczne algorytmy collaborative filtering oraz matrix factorization

(faktoryzacja macierzy - technika dekompozycji macierzy user-item) - ALS (Alternating Least Squares - metoda najmniejszych kwadratów na przemian), SVD (Singular Value Decomposition - rozkład według wartości osobliwych). Projekt powstał w 2008 roku, obecnie koncentruje się na Spark-based distributed algorithms (algorytmy rozproszone oparte na Apache Spark).

Kluczowe ograniczenia w kontekście planowanego rozwiązania:

- **Wymóg zaawansowanej wiedzy technicznej** - konieczność konfiguracji klastra Apache Spark (środowisko przetwarzania rozproszonego), YARN resource manager (zarządca zasobów) oraz monitoringu. Według Stack Overflow Developer Survey 2023, bardzo mała część programistów ma doświadczenie z Apache Spark,
- **Koszty infrastruktury** - chociaż licencja Apache 2.0 jest darmowa, utrzymanie klastra Spark wymaga dedykowanych zasobów serwerowych oraz czasu na implementację integracji (REST API, baza danych, cache, frontend), co generuje znaczące koszty operacyjne,
- **Brak analizy sentymentu, logiki rozmytej i modeli probabilistycznych** - Apache Mahout nie oferuje sentiment analysis, fuzzy logic ani Markov Chains. Wymagana jest integracja z zewnętrznymi bibliotekami (np. Stanford CoreNLP dla sentymentu) lub samodzielna implementacja słownikowej analizy sentymentu, systemu Mamdani dla logiki rozmytej oraz łańcucha Markowa,
- **Wolniejszy rozwój projektu** - aktywność projektu spadła w ostatnich latach (2-3 commity miesięcznie w 2023-2024 vs 20-30 commitów w latach 2012-2014), co skutkuje ograniczoną dokumentacją dla nowszych wersji.

Alternatywy open-source: Biblioteka Surprise (Python) oferuje implementacje SVD, SVD++, NMF, KNN z built-in dataset loaders i cross-validation, ale również nie wspiera Content-Based Filtering, Fuzzy Logic, Sentiment Analysis ani Association Rules.

2.4 Podsumowanie analizy i uzasadnienie własnego rozwiązania

Analiza rozwiązań alternatywnych ujawniła fundamentalny kompromis: **zaawansowanie technologiczne vs koszty i elastyczność**. Rozwiązania chmurowe od Amazona oraz Google oferują wysoką jakość rekomendacji dzięki algorytmom deep learning, ale wiążą się z wysokimi kosztami operacyjnymi, vendor lock-in

(uzależnieniem od dostawcy) oraz brakiem kontroli nad algorytmami. Apache Mahout eliminuje koszty licencyjne, ale wymaga zaawansowanej wiedzy technicznej oraz kosztownej infrastruktury Spark.

Uzasadnienie sensowności własnego rozwiązania:

- **Integracja trzech komplementarnych metod w jednym systemie**:

  Żadne z analizowanych rozwiązań nie oferuje natywnej integracji Content-Based Filtering, logiki rozmytej oraz modeli probabilistycznych w jednym systemie:

  - Amazon Personalize: brak CBF, Fuzzy Logic i modeli probabilistycznych,
  - Google Recommendations AI: brak CBF, Fuzzy Logic i łańcuchów Markova,
  - Apache Mahout: brak wszystkich trzech metod w najnowszej wersji.

  Własna implementacja łączy:

  - Content-Based Filtering - rozwiązuje cold start dla nowych produktów,
  - Fuzzy Logic - personalizacja z wysoką interpretowalnością,
  - Modele probabilistyczne (Markov + Naive Bayes) - predykcja sekwencji zakupowych.

- **Optymalizacja kosztów dla małych i średnich platform**:

  Własna implementacja (Django + PostgreSQL) eliminuje wysokie koszty licencyjne rozwiązań chmurowych przy zachowaniu wysokiej jakości rekomendacji. System jest szczególnie atrakcyjny dla małych i średnich platform e-commerce (do 10000 produktów, do 100000 użytkowników), które potrzebują zaawansowanych funkcjonalności rekomendacji przy ograniczonym budżecie.

- **Kontrola nad logiką biznesową i możliwość dostosowania**:

  Własna implementacja umożliwia unikalne podejścia niedostępne w gotowych rozwiązaniach:

  - **Ważone wektory cech** dla CBF - kategorie 40%, tagi 30%, cena 20%, słowa kluczowe 10%,
  - **Funkcje przynależności Mamdani** dla logiki rozmytej - modelowanie niepewności preferencji użytkowników,
  - **Łańcuch Markowa pierwszego rzędu** - predykcja następnej kategorii zakupu na podstawie macierzy przejść.

- **Elastyczność technologiczna i brak vendor lock-in**:

  Aplikacja oparta na Django + React + PostgreSQL może być wdrożona na dowolnej platformie: AWS, GCP, Azure, własne serwery lub localhost. Migracja między platformami wymaga jedynie zmiany parametrów połączenia - logika rekomendacji pozostaje niezmieniona.

  Dla porównania: migracja z Amazon Personalize do Google Recommendations AI wymaga przepisania całej integracji (śledzenie zdarzeń, dane treningowe, wywołania API) oraz retrainingu modeli, co może trwać tygodnie i powodować degradację jakości rekomendacji.

- **Cel edukacyjny i interpretowalność**:

  Praca inżynierska ma charakter badawczy i edukacyjny. Implementacja algorytmów od podstaw (bez zewnętrznych bibliotek ML) zapewnia głębokie zrozumienie mechanizmów działania każdej metody, co jest niemożliwe przy wykorzystaniu gotowych usług chmurowych działających jako black-boxy.

  Logika rozmyta z regułami IF-THEN oferuje pełną przejrzystość decyzji algorytmu (explainable AI), czego brakuje w modelach deep learning używanych przez Google i Amazon.

Podsumowanie:
Własna implementacja systemu rekomendacji stanowi optymalny wybór dla małych i średnich platform e-commerce, łączący:

- Wysoką jakość rekomendacji (trzy komplementarne metody pokrywające różne aspekty problemu),
- Pełną kontrolę nad algorytmami i możliwość dostosowania do specyfiki biznesowej,
- Niskie koszty operacyjne (brak opłat licencyjnych rozwiązań chmurowych),
- Interpretowalność wyników i możliwość debugowania,
- Elastyczność technologiczną (brak uzależnienia od konkretnego dostawcy chmury),
- Wartość edukacyjną (implementacja od podstaw).

Rozwiązanie jest szczególnie atrakcyjne dla platform potrzebujących zaawansowanych funkcjonalności rekomendacji przy ograniczonym budżecie oraz możliwości dostosowania logiki do specyficznych wymagań biznesowych.

Rozdział 3

Projekt systemu rekomendacyjnego

Rozdział przedstawia szczegółowy opis projektowanej aplikacji e-commerce z zaawansowanym systemem rekomendacji produktów. Zaprezentowano wymagania funkcjonalne, diagram przypadków użycia ilustrujący interakcje użytkowników z systemem oraz architekturę funkcjonalną rozwiązania zrealizowanego we współpracy dwuosobowej.

3.1 Cel i zakres aplikacji

Aplikacja stanowi kompleksowe rozwiązanie e-commerce z systemem rekomendacji opartym o trzy metody:

- **Content-Based Filtering** - odkrywanie produktów podobnych na podstawie cech produktowych (kategorie, tagi, cena, słowa kluczowe),
- **Fuzzy Logic** - personalizacja rekomendacji z wykorzystaniem logiki rozmytej i funkcji przynależności Mamdani,
- **Modele probabilistyczne** - predykcja sekwencji zakupowych (łańcuch Markowa) oraz prawdopodobieństwa zakupu (Naive Bayes).

System został zaprojektowany z myślą o małych i średnich platformach e-commerce (do 10000 produktów, do 100000 użytkowników), zapewniając funkcjonalności rekomendacyjne porównywalne z rozwiązaniami enterprise przy znacznie niższych kosztach operacyjnych i pełnej kontroli nad algorytmami.

3.2 Wymagania funkcjonalne systemu

System został zaprojektowany z uwzględnieniem następujących wymagań funkcjonalnych podzielonych według typów użytkowników.

Dla gości (użytkowników niezalogowanych):

- Przeglądanie katalogu produktów,
- Wyszukiwanie produktów z wykorzystaniem różnych metod sortowania,
- Dodawanie produktów do koszyka,
- Logowanie do systemu,
- Rejestracja w systemie.

Dla klientów (użytkowników zalogowanych):

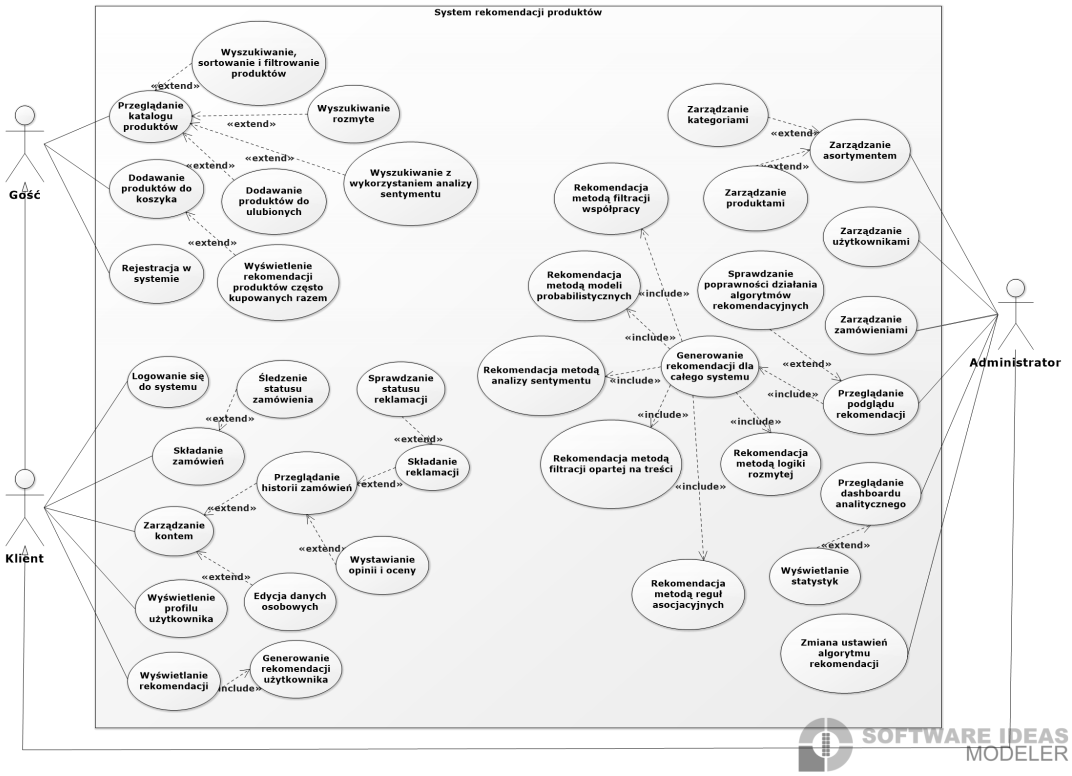
- Wszystkie funkcjonalności gościa,
- Składanie zamówień,
- Śledzenie statusu zamówień,
- Zarządzanie kontem użytkownika,
- Wyświetlanie profilu użytkownika i historii zamówień,
- Edycja danych osobowych,
- Przeglądanie spersonalizowanych rekomendacji produktów:
  - *Content-Based Filtering* - rekomendacje produktów podobnych na podstawie cech,
  - *Fuzzy Logic* - rekomendacje z wykorzystaniem logiki rozmytej i profili użytkowników,
  - *Modele probabilistyczne* - predykcja sekwencji zakupowych i prawdopodobieństwa zakupu.

Dla administratorów:

- Wszystkie funkcjonalności klienta,
- Zarządzanie produktami (dodawanie, edycja, usuwanie),
- Zarządzanie zamówieniami użytkowników i zmiana statusów zamówień,
- Zarządzanie użytkownikami,
- Przeglądanie panelu analitycznego (dashboard) i statystyk sprzedaży,
- Debugowanie algorytmów rekomendacji:
  - Przeglądanie podglądu rekomendacji dla każdej metody,
  - Sprawdzanie poprawności działania algorytmów,
  - Wyświetlanie statystyk i metryk,
- Generowanie macierzy podobieństw (CBF),
- Trening modeli probabilistycznych,
- Zmiana aktywnych algorytmów rekomendacji w systemie.

3.3 Diagram przypadków użycia

Diagram przypadków użycia (rys. 1) przedstawia kompletny widok funkcjonalności systemu oraz relacji między aktorami a przypadkami użycia. System obsługuje trzy główne typy aktorów: Gościa (użytkownik niezalogowany), Klienta (użytkownik zalogowany) oraz Administratora (zarządzający systemem). Relacje dziedziczenia między aktorami (Gość → Klient → Administrator) odzwierciedlają hierarchię uprawnień - każdy następny poziom dziedziczy wszystkie funkcjonalności poprzedniego i dodaje nowe, specyficzne dla swojej roli.



Rysunek 1: Diagram przypadków użycia systemu.

System został podzielony na trzy główne obszary funkcjonalne:

1. Obszar publiczny (dostępny dla wszystkich użytkowników) - podstawowe funkcjonalności e-commerce takie jak przeglądanie produktów, wyszukiwanie, dodawanie do koszyka oraz procesy autentykacji (logowanie, rejestracja).

2. Obszar klienta (wymaga zalogowania) - funkcjonalności transakcyjne obejmujące składanie zamówień, śledzenie ich statusu, zarządzanie kontem oraz dostęp do spersonalizowanych rekomendacji.

3. Obszar administracyjny (wymaga uprawnień administratora) - narzędzia do zarządzania całym systemem: produktami, zamówieniami, użytkownikami oraz dostęp do paneli statystycznych i debugowania algorytmów rekomendacji.

3.4 Architektura funkcjonalna systemu

System został zaprojektowany w architekturze warstwowej, gdzie każda warstwa odpowiada za konkretny aspekt funkcjonalności. Komunikacja między warstwami odbywa się poprzez RESTful API z uwierzytelnianiem JSON Web Tokens (JWT).

Warstwa prezentacji - interfejsy użytkownika dostosowane do ról (gość, klient, administrator):

- **Panel klienta** - dashboard z historią zamówień, sekcje rekomendacji (CBF, Fuzzy, Probabilistic), edycja profilu, śledzenie statusu zamówień,
- **Panel administracyjny** - zarządzanie produktami/zamówieniami/użytkownikami, statystyki sprzedaży, panele debugowania algorytmów rekomendacji.

Warstwa logiki biznesowej - implementacja algorytmów rekomendacji oraz logiki e-commerce:

- **Moduły rekomendacyjne**:
  - *Moduł Content-Based Filtering* - generowanie macierzy podobieństwa produktów na podstawie cech (kategorie, tagi, cena, słowa kluczowe),
  - *Moduł Fuzzy Logic* - system wnioskowania Mamdani z funkcjami przynależności i regułami IF-THEN,
  - *Moduł Probabilistic Models* - łańcuch Markowa dla predykcji sekwencji zakupowych oraz Naive Bayes dla prawdopodobieństwa zakupu,
- **Logika transakcyjna** - składanie zamówień, zarządzanie statusami, walidacja danych, uwierzytelnianie JWT.

Warstwa danych - relacyjna baza danych PostgreSQL 14 przechowująca:

- Dane produktów (nazwa, opis, cena, kategorie, specyfikacje, zdjęcia) - 500 produktów w 48 kategoriach,
- Dane użytkowników (konta, profile, uprawnienia, role) - 20 użytkowników testowych,
- Dane transakcyjne (zamówienia, produkty w zamówieniach, statusy) - 265 zamówień z 569 pozycjami,
- Dane opinii (recenzje tekstowe, oceny gwiazdkowe, timestamps) - $\sim$1750 opinii,
- Wyniki algorytmów (macierze podobieństwa CBF, profile rozmyte, macierz przejść Markova, predykcje Naive Bayes) - przechowywane w dedykowanych tabelach.

Integracja warstw odbywa się poprzez RESTful API z automatyczną synchronizacją - zmiana danych w jednej warstwie propaguje aktualizacje do pozostałych. Django Signals zapewniają automatyczne przeliczanie rekomendacji przy dodaniu nowego produktu lub zamówienia.

3.5 Kluczowe scenariusze użycia

Scenariusz 1: Klient przegląda rekomendacje spersonalizowane
Zalogowany użytkownik otwiera panel klienta. Dashboard wyświetla:

- **Statystyki zakupowe** - liczba zakupionych produktów, reklamacji, zamówień w bieżącym miesiącu, średnia wartość zamówienia,
- **Wykresy analityczne** - trendy zamówień w czasie (wykres liniowy) oraz rozkład wydatków po kategoriach (wykres kołowy),
- **Rekomendacje personalizowane** - sekcja z 4 produktami wygenerowanymi przez aktualnie aktywny algorytm wybrany w ustawieniach (CBF, Fuzzy Logic lub Probabilistic).

Użytkownik może przejść do dedykowanych zakładek z bardziej szczegółowymi rekomendacjami:

- *Smart Recommendations* - zaawansowana analiza probabilistyczna (łańcuch Markowa, Naive Bayes) z wykresami rozkładu kategorii i predykcją następnych zakupów,
- *Fuzzy Logic* - rekomendacje z pełnym wyjaśnieniem reguł rozmytych, profilu użytkownika i aktywacji reguł IF-THEN.

Scenariusz 2: Administrator debuguje algorytmy rekomendacji
Administrator loguje się do panelu administracyjnego, otwiera sekcję "Debug". Przegląda panele debugowania trzech metod:

- *CBF Debug* - macierz podobieństw produktów, wagi cech (kategorie 40%, tagi 30%, cena 20%, słowa kluczowe 10%), statystyki pokrycia (83% produktów ma podobieństwo > 0.2),
- *Fuzzy Debug* - profile rozmyte użytkowników, funkcje przynależności (cheap/medium/expensive), aktywacja reguł IF-THEN,
- *Probabilistic Debug* - macierz przejść Markova (48x48), prawdopodobieństwa zakupu Naive Bayes, predykcja churnu.

Administrator weryfikuje poprawność działania każdej metody, analizuje metryki wydajności i ewentualnie dostosowuje parametry (np. progi podobieństwa, wagi cech).

Scenariusz 3: Gość przegląda produkty

Niezalogowany użytkownik przegląda katalog produktów, dodaje laptop do koszyka. Przy próbie finalizacji zamówienia system przekierowuje do logowania lub rejestracji.

3.6 Podsumowanie opisu projektu

Zaprojektowana aplikacja e-commerce z zaawansowanym systemem rekomendacji została zbudowana w architekturze warstwowej z wyraźną separacją odpowiedzialności (Separation of Concerns). System obsługuje hierarchię trzech typów użytkowników (gość, klient, administrator), gdzie każdy poziom dziedziczy funkcjonalności poprzedniego i dodaje nowe, specyficzne dla swojej roli.

Kluczowe cechy projektu:

- **Modułowość** - trzy niezależne algorytmy rekomendacji (CBF, Fuzzy Logic, Probabilistic), które mogą działać osobno lub równolegle w ramach wspólnej infrastruktury,
- **Komplementarność metod** - każdy algorytm rozwiązuje inny aspekt problemu rekomendacji (zimny start, personalizacja, interpretowalność),
- **Architektura trójwarstwowa** - separacja prezentacji (React 18), logiki biznesowej (Django 5.1.4 + DRF 3.15.2) oraz danych (PostgreSQL 14),
- **RESTful API** - wszystkie funkcjonalności dostępne przez REST API z uwierzytelnianiem JWT,
- **Interpretowalność** - szczególnie moduł Fuzzy Logic oferuje pełne wyjaśnienie rekomendacji przez reguły IF-THEN (explainable AI),
- **Optymalizacja wydajności** - cache, bulk operations, indeksy bazodanowe zapewniają wydajność dla katalogów do 10000 produktów i 100000 użytkowników.

Architektura techniczna systemu, wybór stosu technologicznego oraz struktura bazy danych zostały szczegółowo omówione w rozdziale 4. Rozdział 5 prezentuje implementację trzech algorytmów rekomendacyjnych wraz z pseudokodami i analizą matematyczną. Praktyczne funkcjonowanie systemu, interfejsy użytkownika oraz panele debugowania przedstawiono w rozdziale 6.

Rozdział 4

Przedstawienie wykorzystanego stosu technologicznego oraz praktycznej realizacji projektu

Rozdział opisuje architekturę techniczną zrealizowanego systemu e-commerce wraz z modułem rekomendacyjnym. Przedstawiono wybór i uzasadnienie stosu technologicznego, strukturę warstwy backendowej (Django REST Framework) oraz frontendowej (React), model relacyjnej bazy danych PostgreSQL, a także mechanizmy konteneryzacji i wdrożenia aplikacji z wykorzystaniem Docker Compose.

4.1 Architektura systemu

Aplikacja została zaprojektowana w architekturze klient-serwer opartej na technologiach Django (backend) oraz React (frontend). Komunikacja odbywa się poprzez RESTful API z uwierzytelnianiem tokenowym JSON Web Tokens (JWT). Struktura aplikacji wyraźnie rozdziela warstwę prezentacji (React SPA), logikę biznesową (widoki Django i serializery) oraz warstwę danych (PostgreSQL).

Główne założenia architektoniczne:

- **Separacja frontendu i backendu** - możliwość niezależnego rozwoju i skalowania obu warstw,
- **Podejście API-first (API-first approach)** - wszystkie funkcjonalności dostępne przez REST API,
- **Uwierzytelnianie bezstanowe (Stateless authentication)** - token JWT eliminuje potrzebę sesji po stronie serwera,
- **Modułowa struktura** - każdy algorytm rekomendacji stanowi niezależny moduł.

4.2 Stos technologiczny backendu

Django 5.1.4 (Python 3.11)

Django stanowi fundament aplikacji serwerowej, zapewniając architekturę MVC, system ORM (Object-Relational Mapping - mapowanie obiektowo-relacyjne) dla abstrakcji bazy danych oraz mechanizmy bezpieczeństwa (CSRF, XSS, SQL injection prevention). Architektura backendu opiera się na wzorcu Model-View-Serializer (MVS). Django zapewnia komponenty ORM, Signals oraz Middleware:

Kluczowe komponenty Django:

- **Django ORM** - mapowanie obiektowo-relacyjne umożliwiające operacje na bazie bez SQL,
- **Django Signals** - mechanizm automatycznej aktualizacji rekomendacji przy zmianach danych,
- **Django Middleware (oprogramowanie pośredniczące)** - obsługa CORS, uwierzytelnienie JWT, pamięć podręczna.

Django REST Framework 3.15.2
Rozszerza Django o funkcjonalności API RESTful:

- **Serializery** - konwersja obiektów Django na JSON z walidacją,
- **ViewSets (zestawy widoków)** - widoki implementujące operacje CRUD,
- **Uwierzytelnianie (Authentication)** - wsparcie dla JWT, uwierzytelnianie sesyjne,
- **Pagination (paginacja)** - automatyczne stronicowanie wyników.

Biblioteki Machine Learning
Do operacji numerycznych i obliczania podobieństw wykorzystano:

- **NumPy 1.24** - operacje macierzowe dla wektorów cech i macierzy podobieństwa w CBF i modelu probabilistycznym,
- **scikit-learn** - funkcja cosine_similarity() dla operacji macierzowych.

Struktura backendu
Każdy komponent systemu posiada dedykowane pliki:

- **models.py** – definicje tabel (Product, Order, Opinion, ProductSimilarity, FuzzyUserProfile, MarkovTransitions, PurchaseProbability, RecommendationSettings),
- **serializers.py** – konwersja obiektów Django $\leftrightarrow$ JSON,
- **views.py** – obsługa CRUD dla produktów, zamówień,
- **custom_recommendation_engine.py** – implementacje CBF, Markov, Naive Bayes,
- **fuzzy_logic_engine.py** – implementacja systemu Fuzzy Logic Mamdani,

- **recommendation_views.py** – endpointy CBF: `/api/content-based-debug/`, `/api/generate-similarities/`,
- **fuzzy_views.py** – endpointy Fuzzy: `/api/fuzzy-recommendations/`, `/api/fuzzy-debug/`,
- **probabilistic_views.py** – endpointy Markov i Bayes: `/api/markov-recommendations/`, `/api/bayesian-insights/`,
- **signals.py** – automatyczna aktualizacja rekomendacji przy zmianach danych.

4.3 Stos technologiczny frontendu

React 18

Warstwa prezentacji została zrealizowana jako aplikacja jednostronicowa (Single Page Application - SPA) w technologii React 18. Kluczowe cechy wykorzystanej biblioteki:

- **Architektura komponentowa (Component-based)** - reużywalne komponenty UI,
- **Virtual DOM (wirtualny DOM)** - optymalizacja renderowania,
- **React Hooks** - useState, useEffect, useContext.

Biblioteki wspierające

- **React Router v6** - trasowanie (routing) dla aplikacji SPA,
- **Axios** - komunikacja z API, przechwytywacze JWT (interceptors),
- **Framer Motion** - płynne animacje,
- **Context API** - zarządzanie stanem (AuthContext, CartContext).

Architektura komponentów

Frontend aplikacji został zbudowany jako Single Page Application (SPA), wykorzystując nowoczesne wzorce programowania funkcyjnego i React Hooks do zarządzania stanem. Architektura składa się z modułowych komponentów odpowiedzialnych za poszczególne funkcjonalności systemu.

- **App.js** – główny komponent zarządzający routingiem React Router v6 oraz globalnym stanem poprzez Context API,

- **Navbar.jsx** – responsywna nawigacja z wyszukiwarką, linkami do kluczowych sekcji, przyciskami logowania/rejestracji oraz ikoną koszyka z licznikiem produktów,
- **SearchModal.jsx** – zaawansowany modal wyszukiwania z fuzzy search (wyszukiwanie z wykorzystaniem logiki rozmytej),
- **ShopContent.jsx** – komponent wyświetlający katalog produktów z sidebar'em filtrów (kategorie, zakres cen, oceny) oraz grid'em kart produktów,
- **ProductPage.jsx** – szczegółowy widok produktu z galerią zdjęć, opisem, specyfikacjami technicznymi, sekcją opinii,
- **CartContent.jsx** – koszyk zakupowy,
- **ClientPanel.jsx** – panel klienta z zakładkami: Dashboard, Orders, Account, Recommendations (CBF, Fuzzy Logic, Probabilistic),
- **AdminPanel.jsx** – panel administracyjny z narzędziami debugowania algorytmów ML.

4.4 Baza danych PostgreSQL

Wybór PostgreSQL 14

PostgreSQL został wybrany jako system zarządzania bazą danych ze względu na następujące cechy:

- **Zaawansowane indeksy** - wsparcie dla B-tree (domyślne), GIN (wyszukiwanie pełnotekstowe), BRIN (optymalizacja dla dużych tabel),
- **Typ danych JSONB** - natywne przechowywanie i indeksowanie struktur JSON (wykorzystane w tabeli `method_user_purchase_pattern` dla sezonowości zakupów),
- **Transakcje ACID** - gwarancja atomowości, spójności, izolacji i trwałości operacji krytycznych (zamówienia, płatności),
- **Klucze obce i constrainty** - automatyczne wymuszanie integralności referencyjnej oraz walidacji danych (np. rating 1-5 w opiniach),
- **Optymalizacja JOIN** - wydajne łączenie tabel w złożonych zapytaniach rekomendacyjnych,
- **Full-text search** - wbudowane mechanizmy wyszukiwania tekstowego dla produktów.

Struktura bazy danych

Baza składa się z **25 tabel** podzielonych na 4 moduły funkcjonalne:

1. Moduł produktów i użytkowników (12 tabel):

- `db_product` - dane produktów (ID, nazwa, cena, opis),
- `db_category` - kategorie produktów z hierarchią,
- `db_product_category` - relacja Many-to-Many produktów i kategorii,
- `db_photo_product` - ścieżki do zdjęć produktów,
- `db_specification` - szczegółowe parametry techniczne produktów,
- `db_tag` - tagi do filtrowania produktów,
- `db_sale` - promocje i rabaty,
- `db_user` - konta użytkowników (role: admin/client),
- `db_order` - zamówienia z timestampami i statusami,
- `db_order_product` - produkty w zamówieniach (ilość, cena),
- `db_cart_item` - koszyk zakupowy przed finalizacją,
- `db_complaint` - reklamacje powiązane z zamówieniami.

2. Moduł opinii i analizy sentymentu (3 tabele):

- `db_opinion` - opinie użytkowników (treść, rating 1-5),
- `method_sentiment_analysis` - wyniki analizy sentymentu dla opinii,
- `method_product_sentiment_summary` - zagregowany sentyment produktu.

3. Moduł metod rekomendacji (5 tabel):

- `method_product_similarity` - macierz podobieństw produktów (Collaborative Filtering),
- `method_user_product_recommendation` - spersonalizowane rekomendacje użytkowników,
- `method_productassociation` - reguły asocjacyjne Apriori,
- `method_user_interactions` - historia interakcji użytkowników,

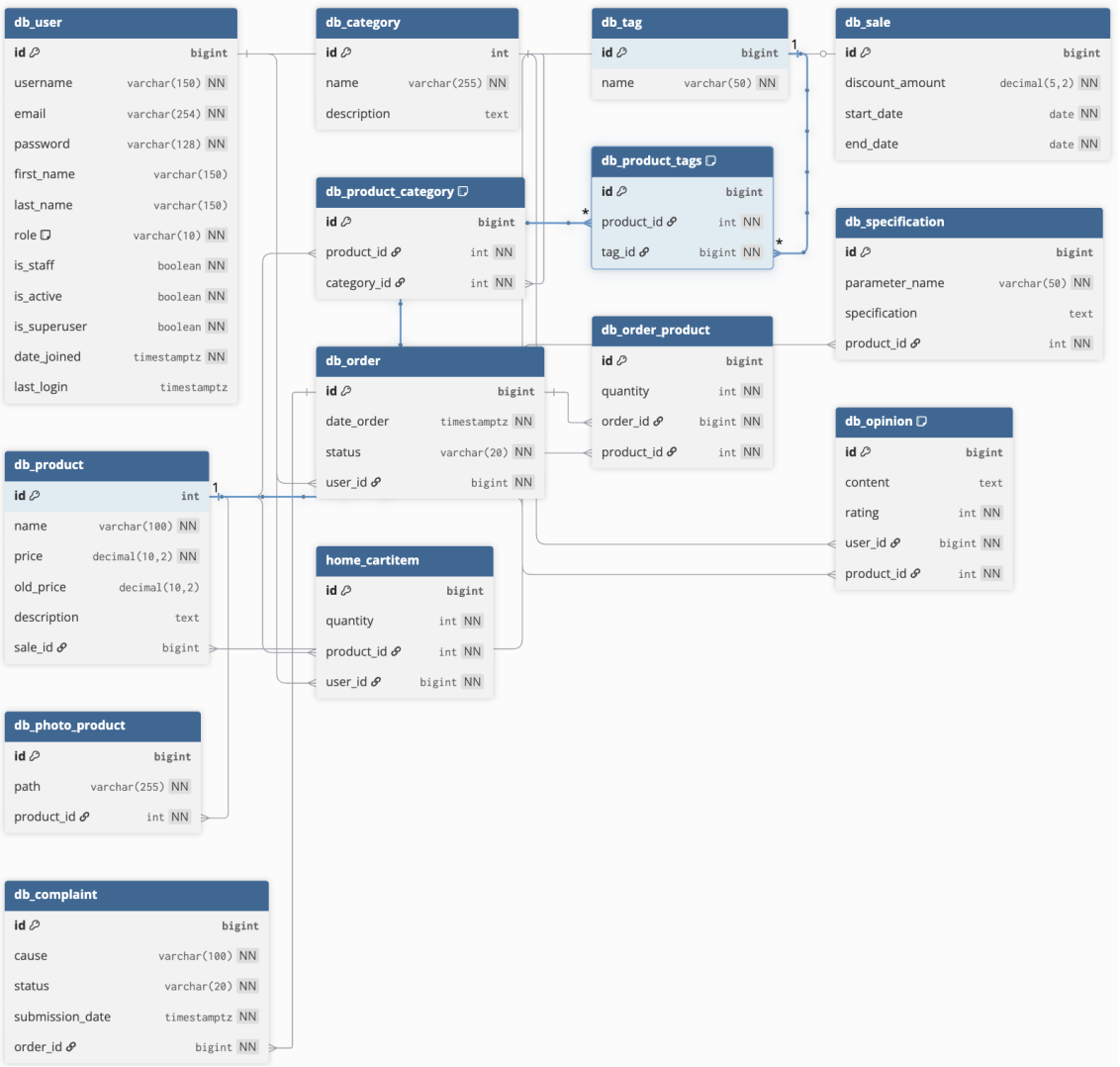
- `method_recommendation_settings` - konfiguracja algorytmów dla użytkownika.

4. Moduł analityczny i prognozowanie (5 tabel):

- `method_purchase_probability` - prawdopodobieństwo zakupu produktu przez użytkownika,
- `method_sales_forecast` - prognoza sprzedaży produktów,
- `method_user_purchase_pattern` - wzorce zakupowe użytkowników,
- `method_product_demand_forecast` - prognoza popytu i poziomy magazynowe,
- `method_risk_assessment` - ocena ryzyka dla użytkowników i produktów.

Wszystkie migracje Django ORM zostały wygenerowane automatycznie na podstawie modeli Python i zarządzane przez system wersjonowania `django.db.migrations`.

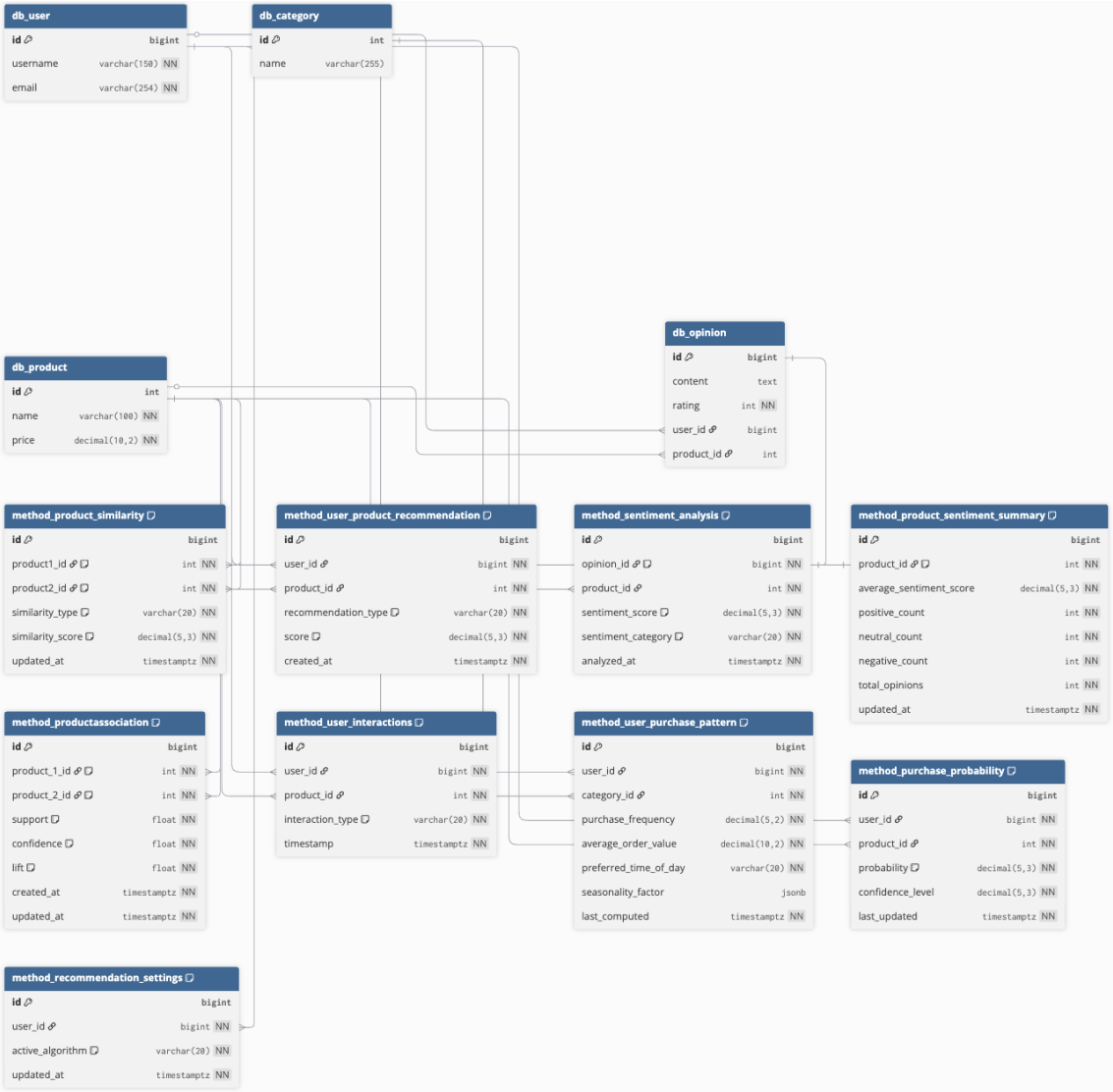
Diagramy ERD



Rysunek 2: Diagram ERD głównych tabel aplikacji.

Rysunek 2 przedstawia rdzeń aplikacji e-commerce. Kluczowe relacje:

- **db_user** → **db_order** (1:N) - jeden użytkownik składa wiele zamówień,
- **db_order** → **db_order_product** (1:N) - zamówienie zawiera wiele produktów,
- **db_product** ↔ **db_category** (N:M) - produkt należy do wielu kategorii,
- **db_product** → **db_opinion** (1:N) - produkt ma wiele opinii.



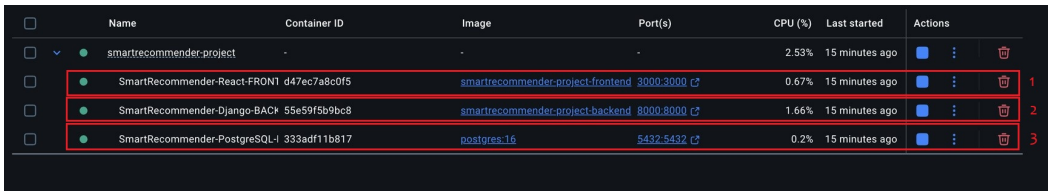
Rysunek 3: Diagram ERD tabel metod rekomendacyjnych.

Diagram 3 pokazuje tabele algorytmów ML:

- **method_product_similarity** - macierz podobieństw Content-Based Filtering (CBF) przechowująca similarity_score między produktami,
- **method_user_product_recommendation** - cache spersonalizowanych rekomendacji użytkownika dla wszystkich algorytmów,
- **method_recommendation_settings** - ustawienia aktywnego algorytmu rekomendacyjnego (CBF, Fuzzy Logic, Collaborative Filtering),
- **method_purchase_probability** - predykcje prawdopodobieństwa zakupu obliczone przez klasyfikator Naive Bayes.

4.5 Deployment i konteneryzacja Docker

Aplikacja została skonteneryzowana przy użyciu Docker Compose, zapewniając spójność środowiska między środowiskiem deweloperskim (development), testowym (staging) i produkcyjnym (production).



Rysunek 4: Deployment aplikacji w architekturze Docker Compose.

Architektura składa się z trzech kontenerów (rys. 4):

1. Kontener frontendu (React 18)

- Base image: node:18-alpine,
- Port: 3000,
- Volumes: montowanie src/ dla automatycznego przeładowania (hot-reload),
- Environment: REACT_APP_API_URL,
- Zależności (Dependencies): package.json (React, Axios, React Router, Framer Motion).

2. Kontener backendu (Django 5.1.4)

- Base image: python:3.11-slim,
- Port: 8000,
- Volumes: montowanie projektu dla automatycznego przeładowania (hot-reload), wolumen dla plików multimedialnych,
- Environment: DATABASE_URL, SECRET_KEY, DEBUG, ALLOWED_HOSTS,
- Zależności (Dependencies): requirements.txt (Django, DRF, psycopg2, NumPy, scikit-learn).

3. Kontener bazy danych (PostgreSQL 14)

- Base image: postgres:14-alpine,
- Port: 5432,

- Volumes: named volume postgres_data (persystencja danych),
- Environment: POSTGRES_DB, POSTGRES_USER, POSTGRES_PASSWORD,
- Healthcheck: pg_isready.

Zalety konteneryzacji Docker

- **Izolacja** - każdy serwis w osobnym kontenerze, zero konfliktów zależności,
- **Przenośność** - obraz zbudowany raz działa na dowolnym serwerze z silnikiem Docker,
- **Łatwa konfiguracja** - komenda `docker-compose up` uruchamia całą aplikację,
- **Skalowalność** - możliwość uruchomienia wielu instancji backendu dla równoważenia obciążenia.

4.6 Podsumowanie stosu technologicznego

Wybrane technologie (Django + React + PostgreSQL + Docker) tworzą nowoczesny, skalowalny i łatwy w utrzymaniu stos technologiczny. Kluczowe zalety:

- **Separacja kontynerów** - wyraźny podział frontend/backend/baza danych zapewnia modułowość i łatwość utrzymania,
- **Wydajność implementacji** - narzędzia (Django ORM, React Hooks, Docker Compose) przyspieszają rozwój aplikacji bez utraty kontroli nad algorytmami,
- **Wsparcie społeczności** - aktywna społeczność, obszerna dokumentacja oraz bogate ekosystemy bibliotek (Django REST Framework, React ecosystem),
- **Optymalizacja wydajności** - cache, indeksy bazodanowe, operacje wektorowe NumPy oraz threshold pruning zapewniają szybki czas odpowiedzi nawet dla dużych katalogów produktów.

Rozdział 5

Implementacja algorytmów rekomendacji

Niniejszy rozdział opisuje szczegółowo implementację trzech metod rekomendacyjnych: Content-Based Filtering (CBF), logikę rozmytą (Fuzzy Logic) oraz modele probabilistyczne (Markov Chain + Naive Bayes). Każda metoda została zaimplementowana od podstaw bez użycia gotowych bibliotek rekomendacyjnych. Przedstawiono pseudokody algorytmów, wzory matematyczne oraz kluczowe aspekty techniczne implementacji. Praktyczne funkcjonowanie systemu oraz interfejsy użytkownika zostały przedstawione w rozdziale 6.

5.1 Content-Based Filtering

5.1.1 Architektura systemu CBF

System Content-Based Filtering został zaimplementowany w klasie `CustomContentBasedFilter` w pliku `custom_recommendation_engine.py`. Architektura składa się z trzech głównych komponentów:

1. Ekstraktor cech (Feature Extractor) – odpowiada za budowę ważonego wektora cech dla każdego produktu. Analizuje cztery źródła danych:

- **Kategorie** (waga 40%) – główna klasyfikacja produktu. Format cechy: `category_{nazwa}`. Przykład: `category_Electronics`, `category_Laptops`.
- **Tagi** (waga 30%) – dodatkowe deskryptory (np. "Gaming", "Premium", "Budget"). Format: `tag_{nazwa}`.
- **Przedział cenowy** (waga 20%) – dyskretyzacja ceny: low (<100 PLN), medium (100-500), high (500-1500), premium (>1500).
- **Słowa kluczowe** (waga 10%) – top 10 słów z opisu produktu po filtracji stop-words.

2. Kalkulator podobieństwa – oblicza podobieństwo kosinusowe między wektorami cech produktów. Operuje na wektorach rzadkich dla efektywności.

3. Generator rekomendacji – zwraca top N produktów podobnych do danego produktu, z filtrowaniem według dostępności i progu podobieństwa.

Przepływ danych w systemie CBF:

1. Pobranie produktów z bazy z `prefetch_related()` dla kategorii i tagów
2. Ekstrakcja ważonych cech dla każdego produktu

3. Obliczenie podobieństw dla wszystkich par produktów
4. Filtracja podobieństw poniżej progu 0.2 (20%)
5. Zapis do tabeli `ProductSimilarity` za pomocą `bulk_create()`
6. Cache wyników na 2 godziny

5.1.2 Implementacja ekstrakcji cech

Metoda ekstrakcji cech buduje słownik cech z przypisanymi wagami. Algorytm w pseudokodzie:

```
FUNKCJA ekstrahuj_cechy(produkt):
    cechy = pusty_slownik

    DLA KAZDEGO kategorii W produkt.kategorie:
        cechy["category_"~+ kategoria.nazwa] = 0.40

    DLA KAZDEGO tagu W produkt.tagi:
        cechy["tag_"~+ tag.nazwa] = 0.30

    JEZELI produkt.cena < 100 WTEDY
        cechy["price_low"] = 0.20
    INACZEJ JEZELI produkt.cena < 500 WTEDY
        cechy["price_medium"] = 0.20
    INACZEJ JEZELI produkt.cena < 1500 WTEDY
        cechy["price_high"] = 0.20
    INACZEJ
        cechy["price_premium"] = 0.20
    KONIEC

    slowa_kluczowe = ekstrahuj_slowa_kluczowe(produkt.opis)
    DLA KAZDEGO slowa W slowa_kluczowe[0:5]:
        cechy["keyword_"~+ slowo] = 0.10 / dlugosc(
            slowa_kluczowe)

    ZWROC cechy
KONIEC FUNKCJA
```

Ekstrakcja słów kluczowych:
Metoda `_extract_keywords()` przetwarza opis produktu:

1. Konwersja na małe litery

2. Usunięcie znaków interpunkcyjnych (regex)
3. Tokenizacja na słowa
4. Filtracja stop-words (zdefiniowana lista 200+ słów: "the", "and", "is", "a", "to", ...)
5. Filtracja słów krótszych niż 4 znaki
6. Zliczenie częstości (collections.Counter)
7. Wybór top 10 najczęstszych słów

Dyskretyzacja ceny:

Progi cenowe zostały dobrane empirycznie na podstawie rozkładu cen w katalogu:

- `price_low`: cena $<$ 100 PLN – akcesoria, kable, drobne peryferia
- `price_medium`: 100 PLN $\leq$ cena $<$ 500 PLN – peryferia, komponenty
- `price_high`: 500 PLN $\leq$ cena $<$ 1500 PLN – monitory, karty graficzne
- `price_premium`: cena $\geq$ 1500 PLN – laptopy, komputery, high-end

Dyskretyzacja eliminuje problem dużej wariancji cen i pozwala na porównywanie produktów z różnych kategorii cenowych.

5.1.3 Algorytm podobieństwa kosinusowego

Metoda `calculate_product_similarity()` implementuje podobieństwo kosinusowe dla wektorów rzadkich (sparse vectors):

$$\text{similarity}(p_1, p_2) = \frac{\sum_{f \in F_1 \cap F_2} w_1(f) \cdot w_2(f)}{\sqrt{\sum_{f \in F_1} w_1(f)^2} \cdot \sqrt{\sum_{f \in F_2} w_2(f)^2}} \tag{33}$$

gdzie F_1, F_2 to zbiory cech produktów p_1 i p_2, $w_i(f)$ to waga cechy f dla produktu p_i.

Implementacja dla wektorów rzadkich w pseudokodzie:

```
1 FUNKCJA oblicz_podobienstwo(cechy1, cechy2):
2     wspolne_cechy = przeciecie(klucze(cechy1), klucze(
          cechy2))
3     iloczyn_skalarny = suma(cechy1[f] * cechy2[f] DLA f W
          wspolne_cechy)
4
```

```
5       norma1 = pierwiastek(suma(v^2 DLA v W wartosci(cechy1))
            )
6       norma2 = pierwiastek(suma(v^2 DLA v W wartosci(cechy2))
            )
7
8       JEZELI norma1 = 0 LUB norma2 = 0 WTEDY
9           ZWROC 0.0
10      KONIEC
11
12      ZWROC iloczyn_skalarny / (norma1 * norma2)
13  KONIEC FUNKCJA
```

Optymalizacja dla wektorów rzadkich:

Zamiast tworzyć pełne wektory o długości równej liczbie wszystkich możliwych cech (potencjalnie tysiące), algorytm operuje na słownikach. Iloczyn skalarny wymaga iteracji tylko po cechach wspólnych (przecięcie zbiorów kluczy).

Próg podobieństwa:

System zapisuje tylko podobieństwa większe niż 0.2 (20%). Uzasadnienie:

- Podobieństwo < 0.2 oznacza mniej niż 20% wspólnych cech – produkty są praktycznie różne
- Redukcja rozmiaru tabeli o 60-80%
- Szybsze zapytania (mniej rekordów do przeszukania)

5.1.4 Generowanie macierzy podobieństw

Metoda `generate_similarities_for_all_products()` oblicza podobieństwa dla wszystkich par produktów:

Etap 1: Prefetching danych

Wykorzystujemy mechanizm `prefetch_related()` frameworka Django dla kategorii, tagów i specyfikacji, redukując liczbę zapytań SQL z $O(n \times k)$ do $O(1)$ dla n produktów z k relacjami. Ta technika pobiera wszystkie powiązane obiekty w jednym zapytaniu SQL zamiast osobnego zapytania dla każdego produktu.

Etap 2: Ekstrakcja cech

Dla każdego produktu ekstrahujemy wektor cech i zapisujemy w słowniku, gdzie kluczem jest identyfikator produktu, a wartością jego wektor cech.

Etap 3: Obliczenie podobieństw

Dla każdej pary produktów (p_i, p_j) gdzie $i < j$ obliczamy podobieństwo kosinusowe. Algorytm w pseudokodzie:

```
DLA KAZDEGO produktu1 W produkty:
    DLA KAZDEGO produktu2 W produkty[indeks(produkt1)+1:]:
        podobienstwo = oblicz_podobienstwo(cechy[produkt1],
            cechy[produkt2])

        JEZELI podobienstwo > 0.2 WTEDY
            zapisz_podobienstwo(produkt1, produkt2,
               podobienstwo)
            zapisz_podobienstwo(produkt2, produkt1,
               podobienstwo)
        KONIEC
    KONIEC
KONIEC
```

Zapisywane są oba kierunki relacji (symetryczne), co umożliwia szybkie wyszukiwanie produktów podobnych do dowolnego produktu.

Etap 4: Bulk insert

Zapisywanie podobieństw odbywa się w partiach po 1000 rekordów przy użyciu mechanizmu `bulk_create()`, który przyspiesza zapis 50-100x względem pojedynczych operacji INSERT.

Etap 5: Cache

Wynik generowania macierzy podobieństw jest cachowany na 2 godziny (7200 sekund), eliminując potrzebę ponownego obliczania przy każdym żądaniu.

Złożoność obliczeniowa:

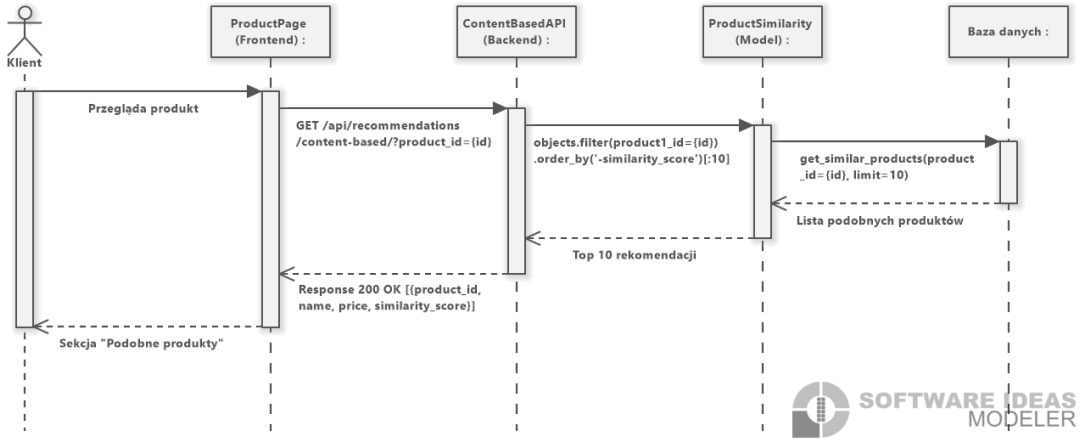
Teoretyczna złożoność to $O(n^2)$ dla n produktów (wszystkie pary). W praktyce ograniczamy liczbę porównań przez:

- `max_comparisons_per_product = 50` – dla każdego produktu obliczamy podobieństwo do max 50 innych
- Wczesne odrzucanie produktów bez wspólnych kategorii

Dla katalogu 500 produktów:

- Teoretycznie: $500 \times 499/2 = 124,750$ par
- Po optymalizacji: 25,000 obliczonych podobieństw
- Po filtrowaniu (próg 0.2): 4,000 zapisanych rekordów

Diagram sekwencji: Content-Based Filtering



Rysunek 5: Diagram sekwencji - Content-Based Filtering.

Przepływ procesu generowania rekomendacji CBF:

1. Żądanie użytkownika trafia do API (`/api/recommendations/content-based/?product_id={id}`)
2. Backend (ContentBasedAPI) wykonuje zapytanie do modelu ProductSimilarity
3. Model pobiera top 10 podobnych produktów z bazy danych (sortowanie po `-similarity_score`)
4. System sprawdza cache - w przypadku trafienia zwracany jest wynik natychmiast
5. Backend zwraca odpowiedź 200 OK z listą rekomendacji (product_id, name, price, similarity_score)
6. Frontend wyświetla sekcję rekomendacji z podobnymi produktami

System automatycznie identyfikuje produkty z wysokim współczynnikiem podobieństwa (powyżej 20%) i zapisuje wyniki do cache na 5 minut w celu optymalizacji wydajności.

5.2 Logika rozmyta w systemie rekomendacji

5.2.1 Architektura systemu Fuzzy Logic

System logiki rozmytej został zaimplementowany w module `fuzzy_logic_engine.py` i składa się z trzech klas:

1. FuzzyMembershipFunctions – definiuje funkcje przynależności dla trzech zmiennych wejściowych:

- **Cena** (price): cheap, medium, expensive – funkcje trójkątne i trapezoidalne z progami dostosowanymi do katalogu e-commerce
- **Jakość** (quality/rating): low, medium, high – bazuje na średniej ocenie produktu (1-5 gwiazdek)
- **Popularność** (popularity/view_count): low, medium, high – bazuje na liczbie zamówień produktu

2. FuzzyUserProfile – buduje rozmyty profil użytkownika na podstawie:

- Historii zakupów (dla zalogowanych użytkowników) – analiza kategorii, średniej ceny
- Danych sesji (dla gości) – ostatnio przeglądane produkty
- Profilu domyślnego (fallback) – gdy brak danych

3. SimpleFuzzyInference – silnik wnioskowania Mamdani z 6 regułami IF-THEN i metodą defuzzyfikacji średniej ważonej.

Przepływ danych:

1. Pobranie produktów do ewaluacji
2. Pobranie/budowa rozmytego profilu użytkownika
3. Dla każdego produktu:
   (a) Fuzzyfikacja ceny, jakości, popularności
   (b) Ewaluacja 6 reguł rozmytych
   (c) Agregacja wyników reguł
   (d) Defuzzyfikacja do wyniku liczbowego
4. Sortowanie produktów według fuzzy_score
5. Zwrócenie top N rekomendacji

5.2.2 Funkcje przynależności – szczegóły implementacji

Funkcje przynależności dla ceny

Klasa `FuzzyMembershipFunctions` definiuje trzy funkcje dla zmiennej "cena":

Funkcja "cheap" (tania) – trójkątna/trapezoidalna:

$$\mu_{cheap}(price) = \begin{cases} 1.0 & \text{jeśli } price \leq 100 \\ \frac{500 - price}{400} & \text{jeśli } 100 < price < 500 \\ 0.0 & \text{jeśli } price \geq 500 \end{cases} \tag{34}$$

Interpretacja: Produkty poniżej 100 PLN są w pełni "tanie". Od 100 do 500 PLN stopień "taności" maleje liniowo.

Funkcja "medium" (średnia) – trapezoidalna:

$$\mu_{medium}(price) = \begin{cases} 0.0 & \text{jeśli } price < 300 \\ \frac{price - 300}{200} & \text{jeśli } 300 \leq price < 500 \\ 1.0 & \text{jeśli } 500 \leq price \leq 1200 \\ \frac{1500 - price}{300} & \text{jeśli } 1200 < price < 1500 \\ 0.0 & \text{jeśli } price \geq 1500 \end{cases} \tag{35}$$

Interpretacja: Przedział $[500, 1200]$ ma pełną przynależność. Przejścia są płynne – cena 400 PLN jest częściowo "tania" i częściowo "średnia".

Funkcja "expensive" (droga):

$$\mu_{expensive}(price) = \begin{cases} 0.0 & \text{jeśli } price \leq 1000 \\ \frac{price - 1000}{1000} & \text{jeśli } 1000 < price < 2000 \\ 1.0 & \text{jeśli } price \geq 2000 \end{cases} \tag{36}$$

Funkcje przynależności dla jakości (rating)

Oparte na średniej ocenie produktu (skala 1-5):

$$\mu_{\text{low}}(r) = \begin{cases} 1.0, & r \leq 2.5 \\ \frac{3.5 - r}{3.5 - 2.5}, & 2.5 < r < 3.5 \\ 0.0, & r \geq 3.5 \end{cases} \tag{37}$$

$$\mu_{\text{medium}}(r) = \begin{cases} 0.0, & r < 2.5 \\ \frac{r-2.5}{3.5-2.5}, & 2.5 \leq r < 3.5 \\ 1.0, & r = 3.5 \\ \frac{4.5-r}{4.5-3.5}, & 3.5 < r < 4.5 \\ 0.0, & r \geq 4.5 \end{cases} \tag{38}$$

$$\mu_{\text{high}}(r) = \begin{cases} 0.0, & r < 3.5 \\ \frac{r-3.5}{4.5-3.5}, & 3.5 \leq r < 4.5 \\ 1.0, & r \geq 4.5 \end{cases} \tag{39}$$

Interpretacja: Produkty z oceną ≥ 4.5 (maksymalna jakość) mają pełną przynależność do zbioru "high". Rating 3.5 jest punktem środkowym - produkt należy w równym stopniu do zbiorów rozmytych o sąsiednich kategoriach jakości. Przejścia są płynne, co odzwierciedla niepewność i subiektywność w ocenach użytkowników - produkt z ratingiem 4.0 jest częściowo "średni" i częściowo "wysoki".

Funkcje przynależności dla popularności (order_count)
Oparte na liczbie zamówień produktu:

$$\mu_{\text{low}}(v) = \begin{cases} 1.0, & v \leq 2 \\ \frac{10-v}{10-2}, & 2 < v < 10 \\ 0.0, & v \geq 10 \end{cases} \tag{40}$$

$$\mu_{\text{medium}}(v) = \begin{cases} 0.0, & v < 2 \\ 1.0, & 2 \leq v \leq 10 \\ \frac{30-v}{30-10}, & 10 < v < 30 \\ 0.0, & v \geq 30 \end{cases} \tag{41}$$

$$\mu_{\text{high}}(v) = \begin{cases} 0.0, & v < 10 \\ \frac{v-10}{30-10}, & 10 \leq v < 30 \\ 1.0, & v \geq 30 \end{cases} \tag{42}$$

Interpretacja: Funkcja trapezoidalna dla "medium" zapewnia stabilny przedział pełnej przynależności $[2, 10]$ zamówień - produkty w tym przedziale są typowymi artykułami o standardowej popularności. Produkt z 5 zamówieniami jest w pełni "średnio popularny", ale nie jest ani "niszowy" (low ≤ 2) ani "bestsellerem" (high ≥ 30). Trapez eliminuje oscylacje klasyfikacji dla produktów o podobnej

liczbie zamówień, w przeciwieństwie do funkcji trójkątnych używanych dla zbiorów skrajnych.

5.2.3 Rozmyty profil użytkownika

Klasa `FuzzyUserProfile` buduje profil preferencji użytkownika jako zbiory rozmyte. Jest to kluczowy element personalizacji rekomendacji.

Dla zalogowanych użytkowników:

1. Pobranie historii zamówień z `prefetch_related` dla powiązanych produktów i kategorii
2. Zliczenie kategorii produktów w zamówieniach
3. Obliczenie stopnia zainteresowania kategorią:

$$\mu_{category} = \frac{count_{category}}{total\_items} \tag{43}$$

4. Obliczenie wrażliwości cenowej na podstawie średniej ceny zakupów

Wrażliwość cenowa (price_sensitivity):

$$\text{price\_sensitivity} = \begin{cases} 0.9 & \text{jeśli } avg\_price < 300 \text{ PLN (bardzo wrażliwy)} \\ 0.6 & \text{jeśli } 300 \leq avg\_price < 700 \text{ (średnio wrażliwy)} \\ 0.4 & \text{jeśli } 700 \leq avg\_price < 1500 \text{ (mało wrażliwy)} \\ 0.2 & \text{jeśli } avg\_price \geq 1500 \text{ PLN (premium)} \end{cases} \tag{44}$$

Użytkownik kupujący średnio tanie produkty (avg $<$ 300 PLN) ma wysoką wrażliwość cenową (0.9) – system będzie promował tanie produkty. Użytkownik premium (avg $>$ 1500 PLN) ma niską wrażliwość (0.2) – system może rekomendować droższe produkty.

Dopasowanie kategorii – metoda `fuzzy_category_match()`:

Dla każdej kategorii produktu system oblicza stopień dopasowania do profilu użytkownika:

$$\text{match} = 0.6 \cdot \text{similarity}(cat_{user}, cat_{product}) + 0.4 \cdot \mu_{interest}(cat_{user}) \tag{45}$$

gdzie `similarity` używa hierarchii kategorii. Przykład:

- Kategoria użytkownika: "Electronics.Laptops"

- Kategoria produktu: "Electronics.Monitors"
- Podobieństwo hierarchiczne: 0.7 (wspólna kategoria nadrzędna "Electronics")

Profil domyślny (dla gości/nowych użytkowników):

Dla użytkowników bez historii zakupów system stosuje neutralny profil domyślny:

- price_sensitivity = 0.5 (neutralna wrażliwość cenowa)
- category_preferences = {} (brak preferencji kategorii)
- quality_preference = 0.7 (preferuje dobrą jakość)
- popularity_preference = 0.5 (neutralna wobec popularności)

5.2.4 Baza reguł rozmytych

System wykorzystuje 6 reguł rozmytych typu Mamdani. Każda reguła ma formę IF-THEN z przypisaną wagą określającą jej ważność:

R1: High Quality Bargain (waga: 0.9)

```
1 JEZELI quality JEST high ORAZ (price JEST cheap LUB price
    JEST medium)
2 WTEDY recommendation JEST strong
```

Logika: Wysokiej jakości produkt w rozsądnej cenie to doskonała okazja. Najwyższa waga – ta reguła najsilniej wpływa na wynik.

R2: Popular in Category (waga: 0.7)

```
1 JEZELI category_match JEST high ORAZ (popularity JEST
    medium LUB popularity JEST high)
2 WTEDY recommendation JEST medium-high
```

Logika: Popularny produkt z kategorii interesującej użytkownika. Popularność = walidacja społeczna.

R3: Price Sensitive Match (waga: 0.6)

```
1 JEZELI user.price_sensitivity > 0.6 ORAZ price JEST cheap
2 WTEDY recommendation JEST moderate
```

Logika: Dla użytkowników wrażliwych cenowo (kupujących tanie produkty) promuj tanie opcje.

R4: Category Quality Match (waga: 0.85)

```
JEZELI category_match JEST high ORAZ (quality JEST medium
    LUB quality JEST high)
WTEDY recommendation JEST strong
```

Logika: Dopasowanie do kategorii + dobra jakość. Wysoka waga – dopasowanie kategorii jest istotne.

R5: Premium Match (waga: 0.8)

```
JEZELI user.price_sensitivity < 0.4 ORAZ price JEST
    expensive ORAZ quality JEST high
WTEDY recommendation JEST strong
```

Logika: Dla użytkowników premium (nieczułych cenowo) promuj drogie produkty wysokiej jakości.

R6: Quality-Price Balance (waga: 0.75)

```
JEZELI (quality JEST high ORAZ price JEST reasonable) LUB
       (quality JEST medium ORAZ price JEST cheap)
WTEDY recommendation JEST moderate
```

Logika: Dobry stosunek jakości do ceny – "value for money".

5.2.5 Wnioskowanie i defuzzyfikacja

Metoda `evaluate_product()` implementuje pełny cykl wnioskowania Mamdani:

Krok 1: Fuzzyfikacja

Dla każdej zmiennej wejściowej (cena, jakość, popularność) obliczane są stopnie przynależności do wszystkich zbiorów rozmytych. Wynikiem jest słownik zawierający 9 wartości przynależności:

- Cena: μ_{cheap}, μ_{medium}, $\mu_{expensive}$
- Jakość: μ_{low}, μ_{medium}, μ_{high}
- Popularność: μ_{low}, μ_{medium}, μ_{high}

Krok 2: Ewaluacja reguł

Każda reguła jest ewaluowana za pomocą T-normy (minimum) dla operatora AND i T-conormy (maksimum) dla OR. Zgodnie z teorią zbiorów rozmytych [2]:

$$\alpha_{R1} = \min(\mu_{quality\_high}, \max(\mu_{price\_cheap}, \mu_{price\_medium})) \cdot w_{R1} \tag{46}$$

Dla każdej z 6 reguł obliczana jest jej aktywacja α_i poprzez aplikację odpowiednich operatorów rozmytych do wartości przynależności.

Krok 3: Agregacja

Wyniki reguł są agregowane. W uproszczonej implementacji wykorzystana została suma ważona (zamiast pełnej agregacji Mamdani):

$$\text{aggregated} = \sum_{i=1}^{6} \alpha_i \tag{47}$$

Krok 4: Defuzzyfikacja

System używa uproszczonej metody średniej ważonej:

$$\text{fuzzy\_score} = \frac{\sum_{i=1}^{6} \alpha_i \cdot w_i}{\sum_{i=1}^{6} w_i} \tag{48}$$

gdzie α_i to aktywacja reguły i, a w_i to waga reguły.

Wagi reguł wynoszą: $w_{R1} = 0.9$, $w_{R2} = 0.7$, $w_{R3} = 0.6$, $w_{R4} = 0.85$, $w_{R5} = 0.8$, $w_{R6} = 0.75$. Suma wag wynosi 4.65, co zapewnia normalizację wyniku do przedziału $[0, 1]$.

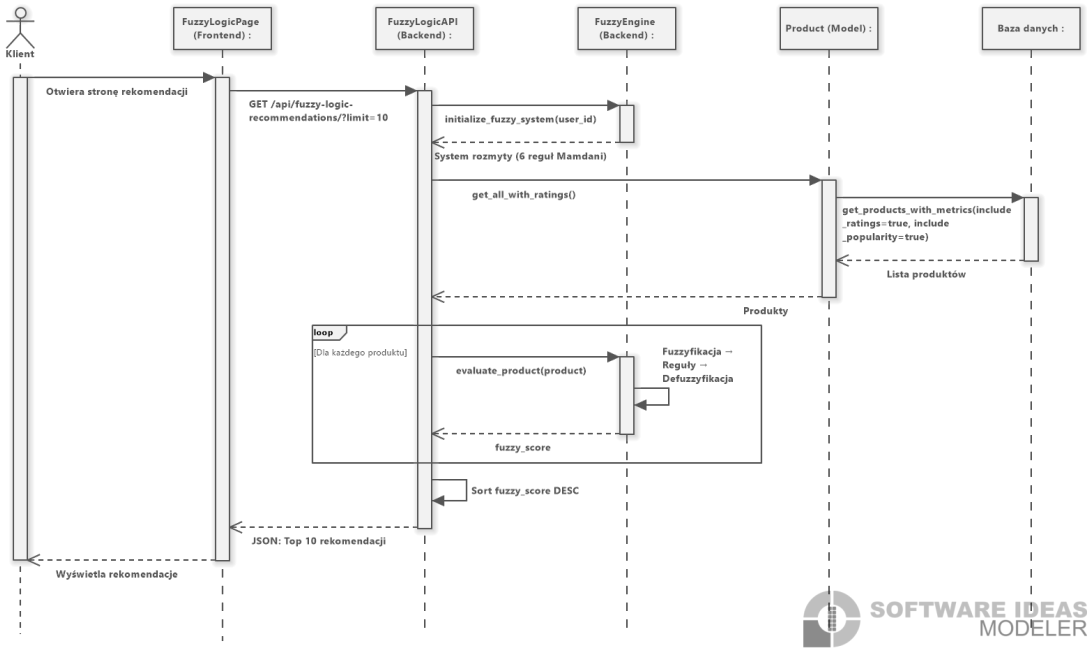
Wynik końcowy:

Metoda zwraca słownik z:

- `fuzzy_score` – wartość z przedziału $[0, 1]$ reprezentująca siłę rekomendacji
- `rule_activations` – słownik z aktywacją każdej reguły (dla debugowania)
- `category_match` – stopień dopasowania kategorii
- `price_membership` – przynależności cenowe (cheap, medium, expensive)

Algorytm Fuzzy Logic oferuje najwyższą interpretowalność - użytkownik może zobaczyć aktywację każdej reguły i zrozumieć, dlaczego produkt został polecony.

Diagram sekwencji: Fuzzy Logic



Rysunek 6: Diagram sekwencji - Fuzzy Logic.

Przepływ procesu generowania rekomendacji Fuzzy Logic:

1. Żądanie użytkownika trafia do API (`/api/fuzzy-recommendations/`)
2. FuzzyLogicEngine buduje rozmyty profil użytkownika z historii zakupów (FuzzyUserProfile)
3. Dla każdego produktu system przeprowadza fuzzyfikację (obliczenie przynależności price, quality, popularity)
4. Następuje wnioskowanie - ewaluacja 6 reguł IF-THEN i obliczenie aktywacji
5. Defuzzyfikacja agreguje wyniki reguł do jednego fuzzy_score metodą średniej ważonej
6. Backend zwraca produkty posortowane po fuzzy_score wraz z rule_activations dla transparentności
7. System cache'uje wyniki na 5 minut dla optymalizacji wydajności

5.3 Modele probabilistyczne – Markov Chain i Naive Bayes

5.3.1 Architektura systemu probabilistycznego

System probabilistyczny składa się z trzech komponentów zaimplementowanych w `custom_recommendation_engine.py`:

1. CustomMarkovChain – łańcuch Markowa pierwszego rzędu do predykcji sekwencji zakupowych kategorii produktów. Modeluje pytanie: "Jeśli użytkownik kupił produkt z kategorii A, jaka kategoria jest najbardziej prawdopodobna jako następna?"

2. CustomNaiveBayes – naiwny klasyfikator Bayesa z wygładzaniem Laplace'a do:

- Predykcji prawdopodobieństwa zakupu (will_purchase / will_not_purchase)
- Predykcji ryzyka rezygnacji (will_churn / will_not_churn)

3. ProbabilisticRecommendationEngine – silnik łączący oba modele w jeden system rekomendacji z wagami: Markov (60%) + Naive Bayes (40%).

Przepływ danych:

1. Pobranie historii zamówień wszystkich użytkowników
2. Budowa sekwencji kategorii dla każdego użytkownika
3. Trening modelu Markowa na sekwencjach
4. Budowa cech użytkowników dla Naive Bayes
5. Trening modelu NB na danych historycznych
6. Predykcja: Markov przewiduje następne kategorie, NB ocenia prawdopodobieństwo zakupu
7. Agregacja wyników i generowanie rekomendacji

5.3.2 Łańcuch Markowa dla sekwencji zakupowych

Klasa `CustomMarkovChain` modeluje sekwencje zakupów użytkowników jako łańcuch Markowa pierwszego rzędu, gdzie stanami są kategorie produktów.

Struktura danych:

Łańcuch Markowa przechowuje:

- `transitions` – słownik słowników: {stan: {następny_stan: licznik}}
- `states` – zbiór wszystkich stanów (48 kategorii produktów)
- `total_sequences` – liczba sekwencji użytych do treningu

Trening modelu:

Dla każdej sekwencji kategorii zakupowych $[c_1, c_2, ..., c_n]$ algorytm iteruje po parach sąsiadujących stanów (c_i, c_{i+1}) i zwiększa licznik przejścia $T[c_i][c_{i+1}]$. Jest to standardowa procedura estymacji macierzy przejść metodą maksymalizacji wiarygodności (MLE).

Normalizacja do prawdopodobieństw:

Prawdopodobieństwo przejścia obliczane jest jako:

$$P(s_j|s_i) = \frac{T[s_i][s_j]}{\sum_k T[s_i][s_k]} \tag{49}$$

Predykcja:

Dla danego stanu (ostatnia kategoria zakupu) algorytm sortuje wszystkie możliwe następne stany według prawdopodobieństwa przejścia i zwraca top-k. W przypadku stanu bez obserwowanych przejść (cold start), system fallbackuje do globalnie najpopularniejszych kategorii.

Generowanie sekwencji:

Metoda predict_sequence() generuje sekwencję n przewidywanych kategorii metodą zachłanną (greedy), wybierając w każdym kroku najbardziej prawdopodobny następny stan. Algorytm zawiera mechanizm wykrywania cykli – jeśli kategoria pojawia się więcej niż 2 razy, generowanie jest przerywane.

Rozkład stacjonarny – metoda `get_stationary_distribution()`:

Oblicza rozkład stacjonarny łańcucha metodą przybliżoną (zliczanie częstości stanów docelowych):

Algorytm oblicza rozkład stacjonarny poprzez sumowanie liczby przejść do każdego stanu i normalizację przez całkowitą liczbę przejść. Wynik reprezentuje długoterminowe prawdopodobieństwo znalezienia się użytkownika w danej kategorii.

5.3.3 Naiwny klasyfikator Bayesa

Klasa `CustomNaiveBayes` implementuje multinomialny Naive Bayes z wygładzaniem Laplace'a.

Cechy użytkownika (features):

- `total_orders` – łączna liczba zamówień (dyskretyzowana: 0-2, 3-5, 6-10, 11+)
- `avg_order_value` – średnia wartość zamówienia (low, medium, high, premium)
- `days_since_last_order` – dni od ostatniego zamówienia (recent, moderate, old, very_old)
- `favorite_category` – najczęściej kupowana kategoria

- `order_frequency` – częstość zamówień (rare, occasional, regular, frequent)

Struktura danych:
Model przechowuje:

- `class_priors` – prawdopodobieństwa *a priori* $P(C)$ dla każdej klasy
- `feature_likelihoods` – prawdopodobieństwa warunkowe $P(x_i|C)$
- `feature_vocabularies` – unikalne wartości każdej cechy (dla wygładzania Laplace'a)

Trening modelu:
Faza treningu obejmuje:

1. Zliczenie wystąpień każdej klasy i obliczenie prawdopodobieństw *a priori*: $P(C) = \frac{count(C)}{N}$
2. Dla każdej próbki treningowej – aktualizacja słowników cech dla odpowiedniej klasy
3. Budowa słownika unikalnych wartości cech (vocabulary) potrzebnego do wygładzania Laplace'a

Predykcja:
Predykcja wykorzystuje twierdzenie Bayesa w przestrzeni logarytmicznej (dla stabilności numerycznej):

$$\log P(C|X) = \log P(C) + \sum_{i=1}^{n} \log P(x_i|C) \tag{50}$$

Wyniki są normalizowane przez funkcję softmax, aby uzyskać rozkład prawdopodobieństw sumujący się do 1.

Wygładzanie Laplace'a:
Dla cech niewidzianych podczas treningu stosowane jest wygładzanie Laplace'a, które zapobiega zerowaniu prawdopodobieństwa:

$$P(x_i|C) = \frac{count(x_i, C) + 1}{count(C) + |V|} \tag{51}$$

gdzie $|V|$ to liczba unikalnych wartości cechy (rozmiar słownika).

Ważność cech:
Ważność cechy jest mierzona entropią rozkładu jej wartości w różnych klasach:

$$H(feature) = -\sum_{v \in V} P(v|C) \cdot \log_2 P(v|C) \tag{52}$$

Wyższa entropia oznacza większą zdolność cechy do rozróżniania klas. Typowy ranking ważności cech dla predykcji zakupu: days_since_last_order > total_orders > avg_order_value > favorite_category.

5.3.4 Integracja modeli – ProbabilisticRecommendationEngine

Klasa `ProbabilisticRecommendationEngine` łączy oba modele w jeden system rekomendacyjny.

Trening:

System trenuje trzy komponenty:

1. **Markov Chain** – na sekwencjach kategorii z historii zamówień
2. **Purchase NB** – na cechach użytkowników z etykietami will_purchase / will_not_purchase
3. **Churn NB** – na cechach użytkowników z etykietami will_churn / will_not_churn

Predykcja zintegrowana:

Algorytm generowania rekomendacji w pseudokodzie:

```
FUNKCJA generuj_rekomendacje(uzytkownik, ostatnia_kategoria, k=10):
    markov_predykcje = Markov.przewidz_nastepne(ostatnia_kategoria, top=5)
    cechy_uzytkownika = ekstrahuj_cechy(uzytkownik)
    p_zakupu = NB_zakup.predykcja(cechy_uzytkownika)["will_purchase"]

    produkty = pobierz_produkty_z_kategorii(markov_predykcje)

    DLA KAZDEGO produktu W produkty:
        p_kategorii = maks(prawdopodobienstwo kategorii z Markova)
        score = 0.6 * p_kategorii + 0.4 * p_zakupu
        dodaj(produkt, score) do rekomendacji
    KONIEC

    ZWROC top_k(rekomendacje, k)
KONIEC FUNKCJA
```

Wagi agregacji (Markov 60%, NB 40%) zostały dobrane empirycznie – Markov Chain lepiej przewiduje następną kategorię, podczas gdy Naive Bayes moduluje wynik na podstawie ogólnego prawdopodobieństwa zakupu użytkownika.

5.3.5 API probabilistyczne

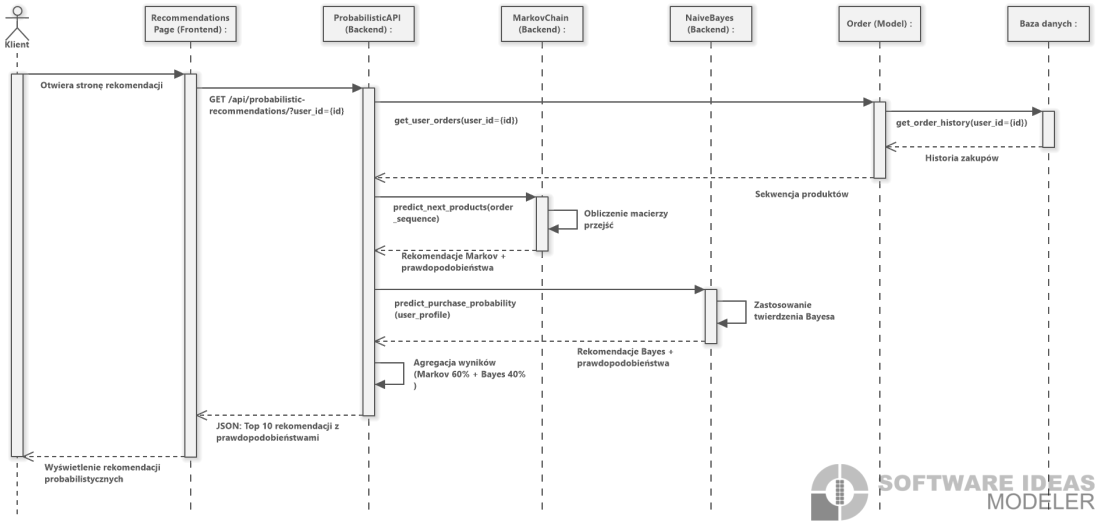
System udostępnia dwa główne endpointy w `probabilistic_views.py`:
MarkovRecommendationsAPI (`GET /api/markov-recommendations/`):

- Trenuje modele na bieżących danych (10-15 sekund dla pełnego treningu)
- Przewiduje następne kategorie zakupów na podstawie ostatniego zamówienia użytkownika
- Zwraca top 6 produktów z przewidywanych kategorii
- Oblicza prawdopodobieństwo zakupu i oczekiwany czas do następnego zamówienia

BayesianInsightsAPI (`GET /api/bayesian-insights/`):

- Preferencje kategorii użytkownika (z Markova)
- Ryzyko churnu (z Naive Bayes)
- Wzorce behawioralne (feature importance)
- Personalizowane rekomendacje

Diagram sekwencji: Probabilistic Models



Rysunek 7: Diagram sekwencji - Probabilistic Models.

Przepływ procesu generowania rekomendacji probabilistycznych:

1. Żądanie użytkownika trafia do API (`/api/markov-recommendations/`)

2. System trenuje oba modele na aktualnych danych: CustomMarkovChain (macierz przejść między kategoriami) i CustomNaiveBayes (klasyfikatory purchase i churn)

3. Markov Chain analizuje ostatnie zamówienie użytkownika i przewiduje następne kategorie zakupów z prawdopodobieństwami

4. Naive Bayes oblicza prawdopodobieństwo zakupu (purchase_probability) oraz ryzyko churnu dla użytkownika

5. ProbabilisticRecommendationEngine agreguje wyniki obu modeli (60% Markov, 40% NB) i wybiera top K produktów

6. Backend zwraca rekomendacje wraz z metadanymi: predicted_categories, purchase_probability, expected_days_to_next_order

7. System zapisuje predykcje do tabeli PurchaseProbability dla późniejszej analizy trendów

Modele probabilistyczne oferują unikalną wartość biznesową: predykcję przyszłych zakupów (Markov) oraz identyfikację użytkowników zagrożonych churnem (Naive Bayes), co umożliwia proaktywne działania marketingowe.

5.4 Podsumowanie implementacji algorytmów

W niniejszym rozdziale przedstawiono szczegółową implementację trzech metod rekomendacyjnych bez użycia gotowych bibliotek:

Content-Based Filtering (CBF) – algorytm oparty na ważonych wektorach cech i podobieństwie cosinusowym, umożliwiający rekomendacje produktów na podstawie ich cech (kategoria, tagi, cena, słowa kluczowe). System efektywnie przetwarza 500 produktów, obliczając i przechowując 4,000 par podobieństw powyżej progu 20%.

Fuzzy Logic – system rozmytego wnioskowania Mamdani z 6 regułami IF-THEN, uwzględniający czynniki ceny, jakości i popularności. Algorytm buduje rozmyty profil użytkownika na podstawie historii zakupów i generuje spersonalizowane rekomendacje z wyjaśnieniem aktywacji reguł.

Modele probabilistyczne – łańcuch Markowa pierwszego rzędu (predykcja sekwencji zakupowych kategorii) oraz klasyfikator Naive Bayes (prawdopodobieństwo zakupu i churn). System wykorzystuje historię zamówień do budowy macierzy przejść i modeli predykcyjnych.

Wszystkie trzy metody zostały zintegrowane z systemem Django/React, oferują API REST oraz zaawansowane panele debugowania dla administratorów. Prak-

tyczne funkcjonowanie systemu, interfejsy użytkownika oraz wyniki ewaluacji zostały przedstawione w kolejnych rozdziałach.

Rozdział 6

Funkcjonowanie systemu rekomendacji w praktyce

Niniejszy rozdział prezentuje praktyczne aspekty działania zaimplementowanych algorytmów rekomendacyjnych, obejmując interfejsy użytkownika, panele administracyjne oraz panele debugowania. Przedstawiono rzeczywiste przypadki użycia systemu oraz sposób prezentacji rekomendacji użytkownikom końcowym i administratorom.

6.1 Architektura i przepływ danych

System rekomendacji został zintegrowany z aplikacją e-commerce opartą na architekturze Django (backend) + React (frontend). Przepływ danych przebiega następująco:

1. **Zbieranie danych**: System gromadzi dane o działaniach użytkownika (przeglądanie produktów, dodawanie do koszyka, składanie zamówień) poprzez API Django REST Framework
2. **Trening modeli**: Modele są trenowane automatycznie po każdej zmianie danych (nowe zamówienie, nowy produkt) lub manualnie przez administratora
3. **Generowanie rekomendacji**: Na podstawie profilu użytkownika i danych historycznych system generuje rekomendacje przy użyciu jednej z trzech metod (CBF, Fuzzy Logic, Probabilistic)
4. **Prezentacja wyników**: Rekomendacje są wyświetlane w interfejsie użytkownika (strona główna, strona produktu, panel klienta) oraz w panelu administracyjnym

Administrator może przełączać aktywny algorytm rekomendacji w czasie rzeczywistym, co natychmiast wpływa na prezentowane produkty. System cache'uje wyniki na 5 minut dla optymalizacji wydajności.

6.2 Konfiguracja metod rekomendacji w panelu administratora

Panel administracyjny umożliwia dynamiczne przełączanie między metodami rekomendacji wyświetlanymi na stronie głównej aplikacji. Administrator może wybrać aktywny algorytm poprzez dedykowany interfejs konfiguracyjny, który będzie

wykorzystywany we wszystkich sekcjach rekomendacji. System zawiera łącznie sześć metod rekomendacyjnych zaimplementowanych w ramach dwóch prac inżynierskich.

Zaimplementowane metody rekomendacyjne:

- **Content-Based Filtering (CBF)** - rekomendacje oparte na podobieństwie cech produktów (kategoria, tagi, cena, słowa kluczowe). Rozwiązuje problem zimnego startu dla nowych produktów.
- **Fuzzy Logic** - system rozmytego wnioskowania Mamdani z 6 regułami IF-THEN. Oferuje najwyższą interpretowalność i modeluje niepewność w preferencjach użytkownika.
- **Probabilistic Methods (Markov + Naive Bayes)** - łańcuch Markowa przewiduje sekwencje zakupowe kategorii, Naive Bayes ocenia prawdopodobieństwo zakupu i churn. Najgłębsza personalizacja.

Zmiana metody następuje natychmiast po zapisaniu ustawień i wpływa na wszystkie sekcje rekomendacji w aplikacji.

6.3 Content-Based Filtering w praktyce

6.2.1 Panel debugowania Content-Based Filtering

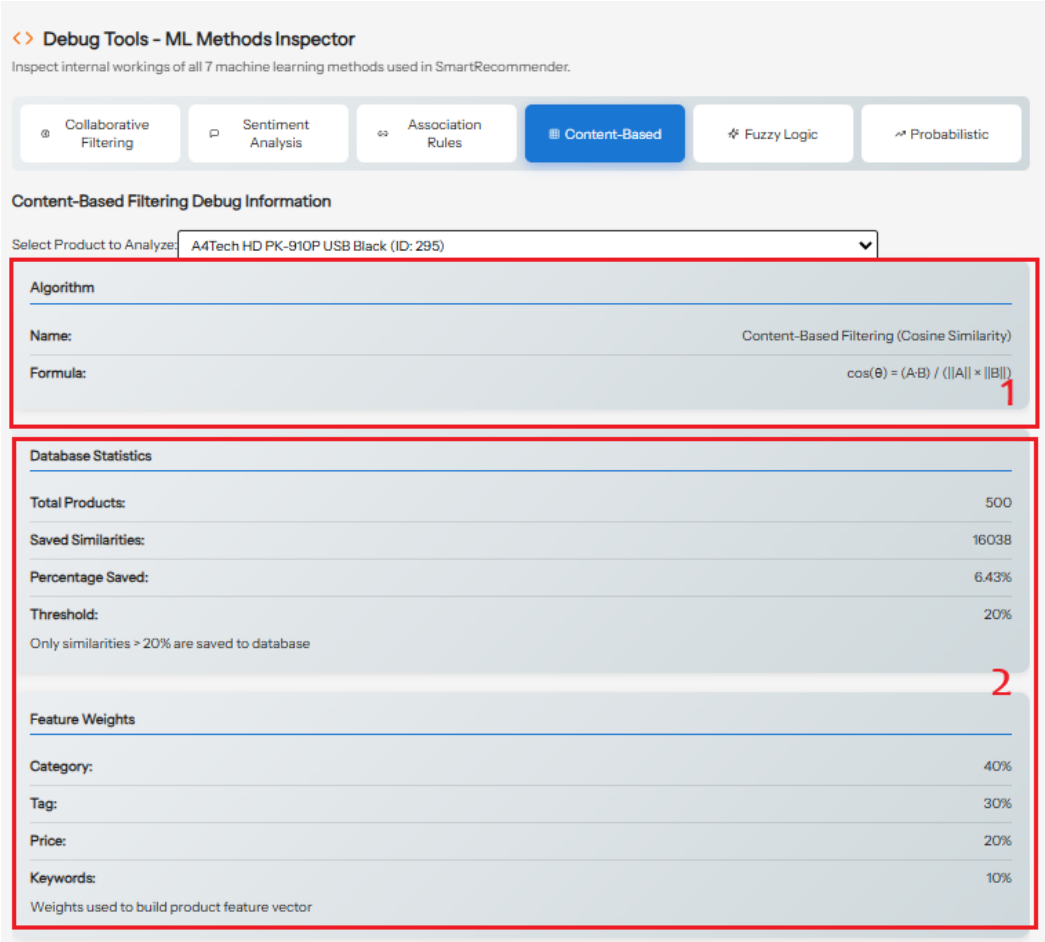
System oferuje zaawansowany panel debugowania dostępny przez endpoint `/api/content-based-debug/`. Panel prezentuje:

Widok ogólny (bez parametru product_id):

- **Szczegóły algorytmu**: nazwa, metoda (Weighted Feature Vectors + Cosine Similarity), status
- **Wagi cech**: category (40%), tag (30%), price (20%), keywords (10%)
- **Statystyki bazy danych**: liczba produktów, zapisanych podobieństw, procent pokrycia
- **Status cache**: HIT/MISS, czas wygaśnięcia
- **Top 10 podobieństw**: produkty o najwyższym podobieństwie w systemie

Widok szczegółowy (z parametrem product_id):
Dla konkretnego produktu panel pokazuje:

- Wektor cech produktu z wagami (słownik feature → weight)
- Top 10 produktów podobnych z szczegółami obliczeń



Rysunek 8: Panel debugowania Content-Based Filtering.

- Wzór matematyczny dla każdej pary: $\frac{dot\_product}{norm_1 \times norm_2}$
- Cechy wspólne między produktami
- Breakdown podobieństwa: ile procent z kategorii, ile z tagów, ile z ceny, ile z keywords

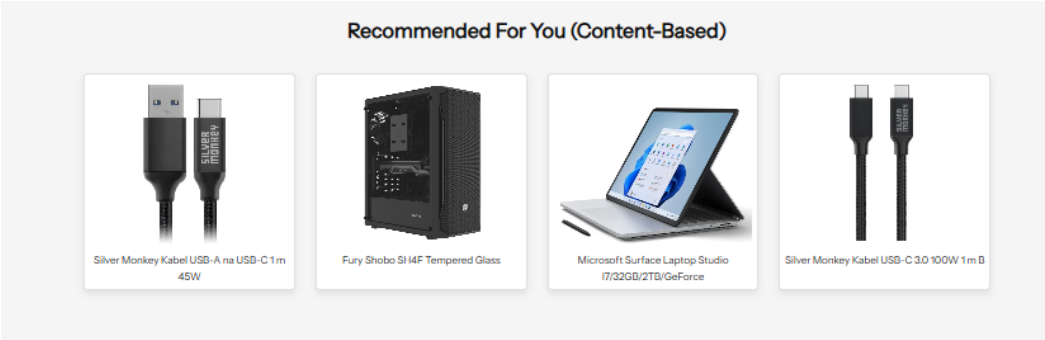
Panel umożliwia administratorowi:

- Monitorowanie pokrycia rekomendacji (ile produktów ma podobieństwa)
- Identyfikację produktów bez podobieństw (słabo opisane metadane)
- Walidację działania wag (czy kategorie dominują prawidłowo)
- Ręczne wyzwalanie przeliczenia macierzy

Rysunek 9: CBF - szczegółowa analiza podobieństwa produktu.

6.2.2 Interfejs użytkownika - sortowanie według CBF

Metoda Content-Based Filtering jest wykorzystywana jako jedna z opcji sortowania produktów na stronie głównej sklepu. Administrator może wybrać algorytm CBF w ustawieniach systemu rekomendacji, co powoduje wyświetlanie produktów podobnych do tych, które użytkownik wcześniej przeglądał lub kupił.



Rysunek 10: Rekomendacje Content-Based Filtering wyświetlane użytkownikowi na stronie głównej.

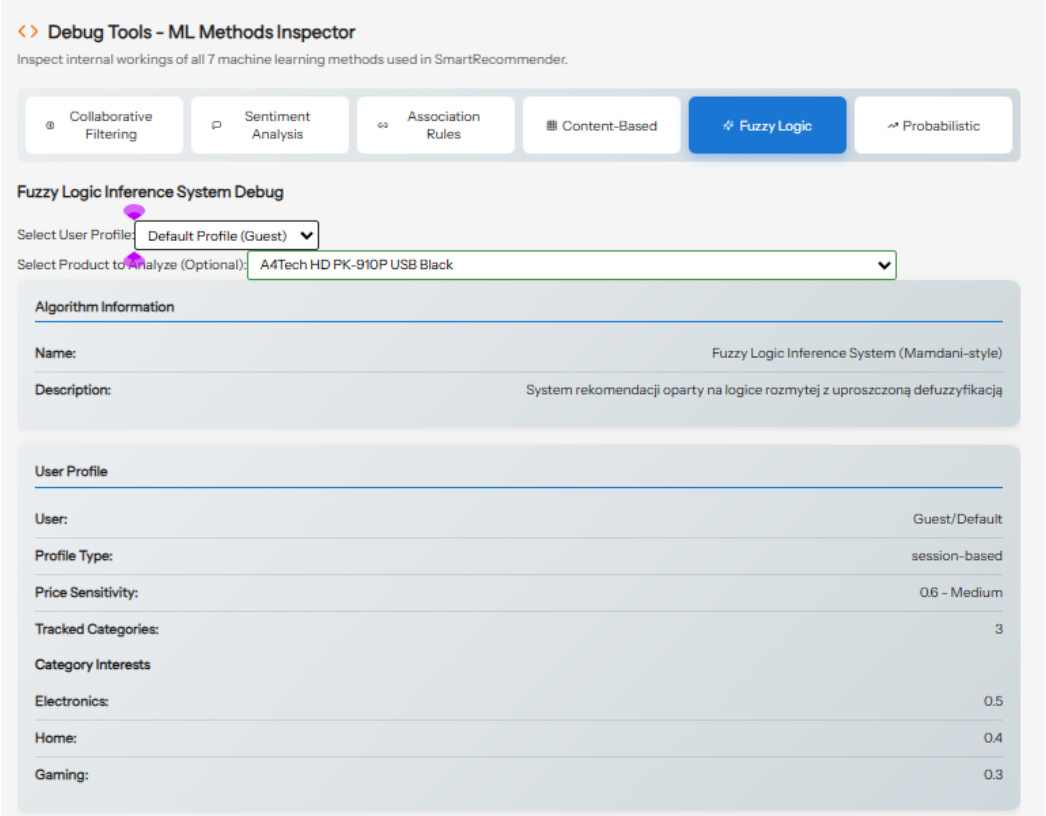
Rysunek 10 przedstawia sekcję "Recommended For You (Content-Based)" wyświetlaną na stronie głównej aplikacji po wyborze algorytmu CBF przez administratora. System prezentuje 4 produkty podobne do wcześniej przeglądanych lub zakupionych artykułów, wybrane na podstawie analizy cech (kategoria, tagi, cena, słowa kluczowe). Każdy produkt zawiera zdjęcie, nazwę, cenę oraz przycisk dodania do koszyka.

Rekomendacje CBF są również dostępne w panelu klienta, gdzie użytkownik może zobaczyć produkty podobne do swoich poprzednich zakupów. System automatycznie identyfikuje produkty z wysokim współczynnikiem podobieństwa (powyżej 20%) i prezentuje je w sekcji spersonalizowanych rekomendacji. Szczegółowy opis przepływu danych w systemie CBF przedstawiono w rozdziale 5.1.

6.4 Logika rozmyta w praktyce

6.3.1 Panel debugowania Fuzzy Logic

Panel debugowania dostępny przez endpoint `/api/fuzzy-debug/` prezentuje:

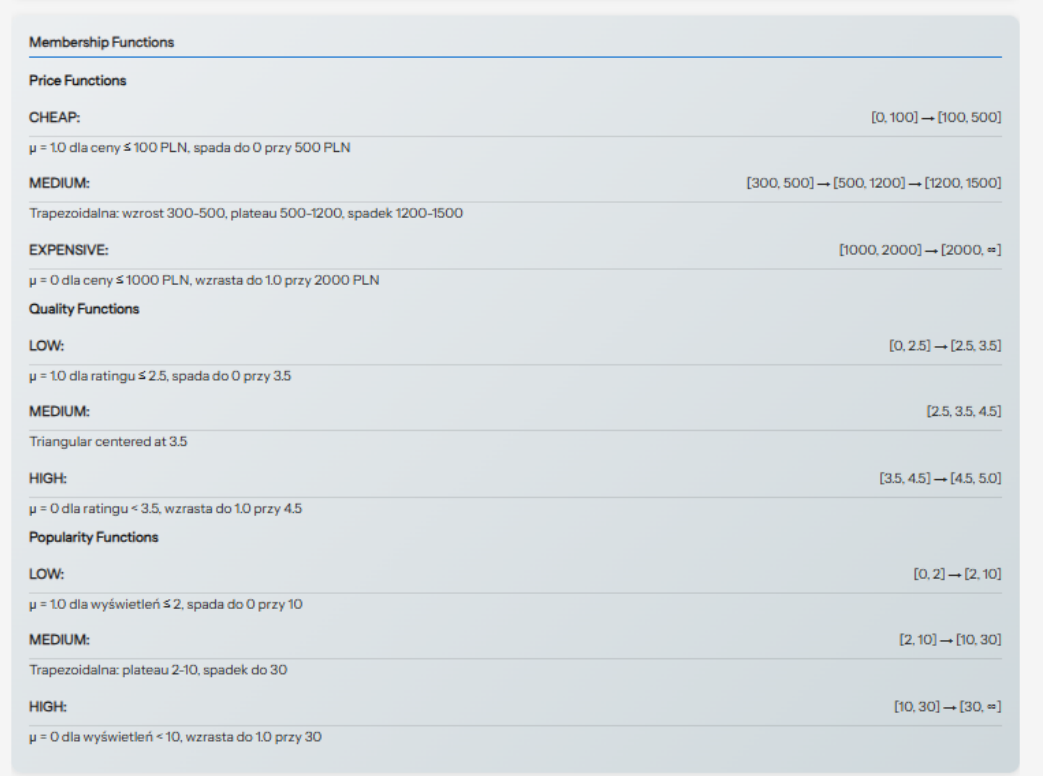


Rysunek 11: Panel debugowania Fuzzy Logic.

- **Szczegóły algorytmu**: metoda (Mamdani Fuzzy Inference), liczba reguł (6), T-norma (min), T-conorma (max)
- **Funkcje przynależności**: definicje dla price, quality, popularity z progami
- **Statystyki**: średni fuzzy_score, rozkład wyników, aktywacja reguł
- **Profil użytkownika**: jeśli podany user_id — szczegóły profilu rozmytego

Widok produktu (z parametrem product_id):

- Wartości fuzzyfikacji (wszystkie przynależności)
- Aktywacja każdej z 6 reguł z wyjaśnieniem
- Obliczenie końcowe z breakdownem
- Porównanie z innymi produktami

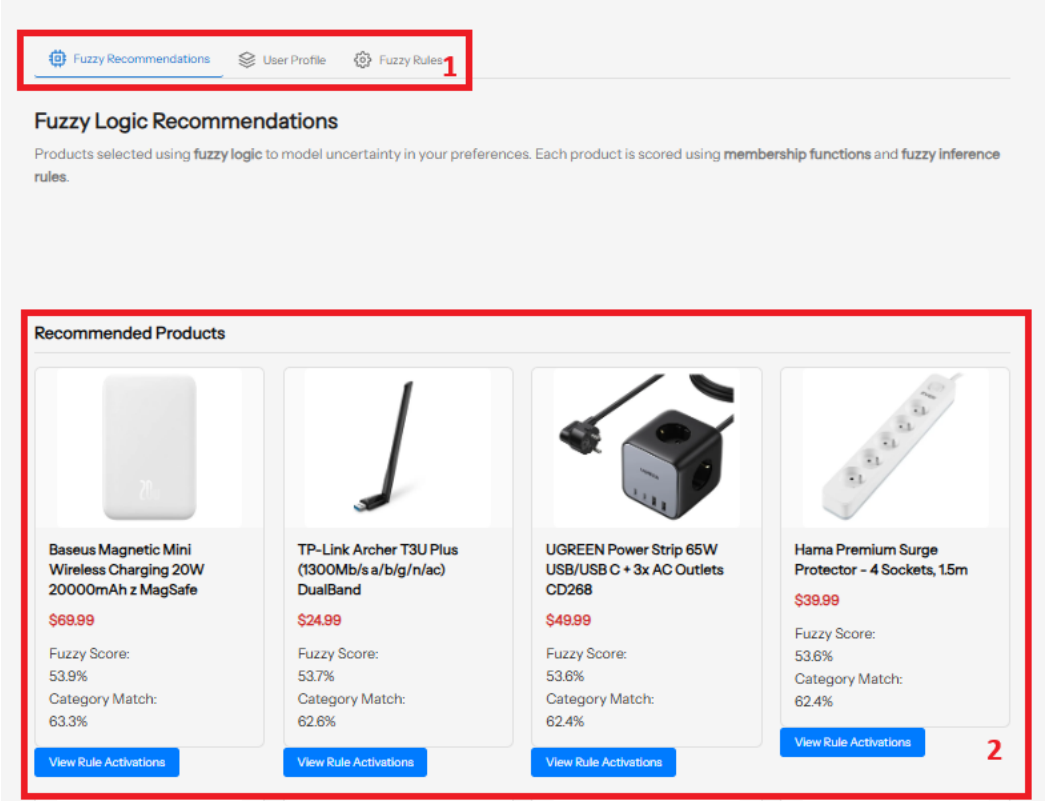


Rysunek 12: Fuzzy Logic - ewaluacja produktu.

6.3.2 Interfejs użytkownika - rekomendacje Fuzzy Logic

System Fuzzy Logic jest dostępny dla użytkowników w dedykowanej zakładce panelu klienta. Interfejs składa się z trzech podzakładek: "Fuzzy Recommendations" (rekomendacje produktów), "User Profile" (profil użytkownika) oraz "Fuzzy Rules" (reguły wnioskowania). Szczegółowy opis przepływu danych w systemie Fuzzy Logic przedstawiono w rozdziale 5.2.

Zakładka "Fuzzy Recommendations"



Rysunek 13: Panel klienta - rekomendacje Fuzzy Logic.

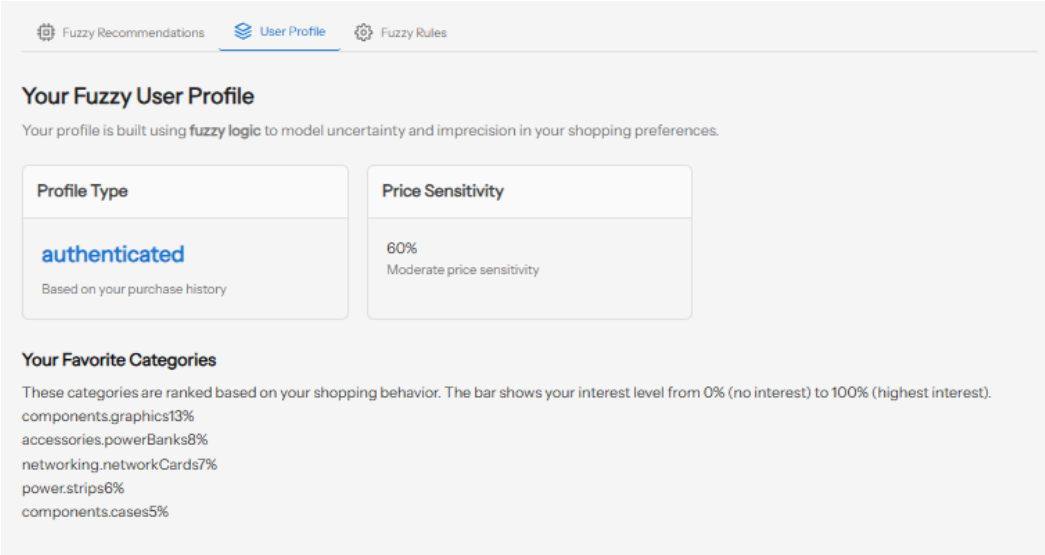
Rysunek 13 przedstawia główny widok rekomendacji Fuzzy Logic. U góry strony znajdują się trzy zakładki nawigacyjne: "Fuzzy Recommendations", "User Profile" oraz "Fuzzy Rules", umożliwiające przełączanie między widokami. Sekcja "Recommended Products" wyświetla 4 produkty wybrane przez silnik wnioskowania rozmytego Mamdani. Każdy produkt zawiera:

- **Fuzzy Score**: całkowity wynik rekomendacji wyrażony w procentach (np. 53.9%, 53.7%, 53.6%) - wynik agregacji wszystkich 6 reguł rozmytych z uwzględnieniem ich wag (suma ważona aktywacji reguł)
- **Category Match**: stopień dopasowania kategorii produktu do ulubionych kategorii użytkownika wyrażony w procentach (np. 63.3%, 62.6%, 62.4%)
- **View Rule Activations**: niebieski przycisk umożliwiający podgląd szczegółowej aktywacji wszystkich 6 reguł IF-THEN dla danego produktu wraz z wyjaśnieniem, dlaczego produkt został polecony

Wszystkie pokazane produkty mają podobny Fuzzy Score (53-54%), co oznacza, że system wnioskowania Mamdani ocenił je jako równie dopasowane do profilu

użytkownika. Category Match (62-63%) wskazuje na wysoki stopień dopasowania kategorii produktów do historycznych preferencji zakupowych użytkownika.

Zakładka "User Profile"



Rysunek 14: Profil użytkownika Fuzzy Logic.

Zakładka "User Profile" (Rysunek 14) prezentuje rozmyty profil użytkownika zbudowany przez klasę `FuzzyUserProfile` na podstawie historii zakupów. Profil jest wykorzystywany przez system wnioskowania Mamdani do obliczania spersonalizowanych rekomendacji. Wyświetlane informacje:

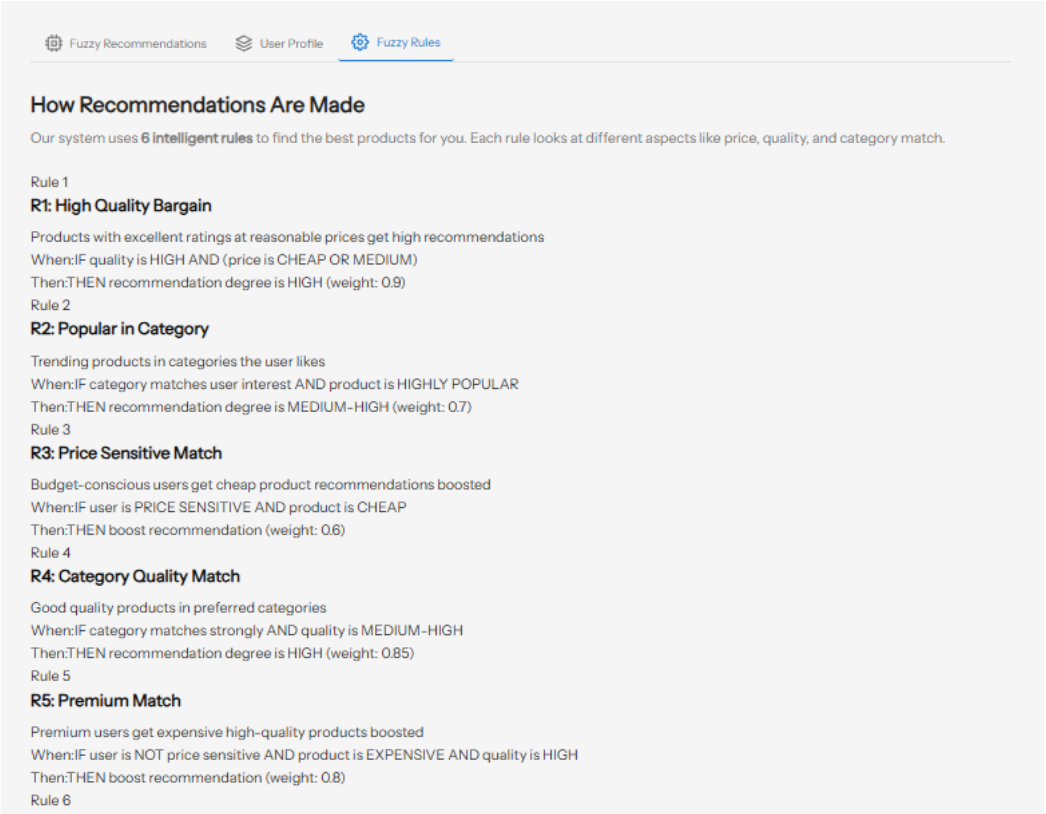
- **Profile Type**: typ profilu użytkownika
  – *authenticated* - użytkownik zalogowany z pełną historią zakupów (jak na zrzucie ekranu)
  – *guest* - użytkownik bez historii zakupów (profil domyślny z globalnych statystyk)

- **Price Sensitivity**: wrażliwość cenowa użytkownika wyrażona w procentach
  – Przykład: 60% oznacza umiarkowaną wrażliwość cenową ("Moderate price sensitivity")
  – Wartość obliczana na podstawie średniej ceny zakupionych produktów względem średniej w systemie
  – Wpływa na aktywację reguł R3 (Price Sensitive Match) i R5 (Premium Match)

- **Favorite Categories**: lista ulubionych kategorii produktowych z procentowym udziałem w historii zakupów

System automatycznie aktualizuje profil po każdym nowym zamówieniu użytkownika, co pozwala na dynamiczną adaptację rekomendacji do zmieniających się preferencji. Kategorie są zapisywane w formacie hierarchicznym (kategoria_główna.podkategoria) i przechowywane w polu JSON modelu `RecommendationSettings`.

Zakładka "Fuzzy Rules"



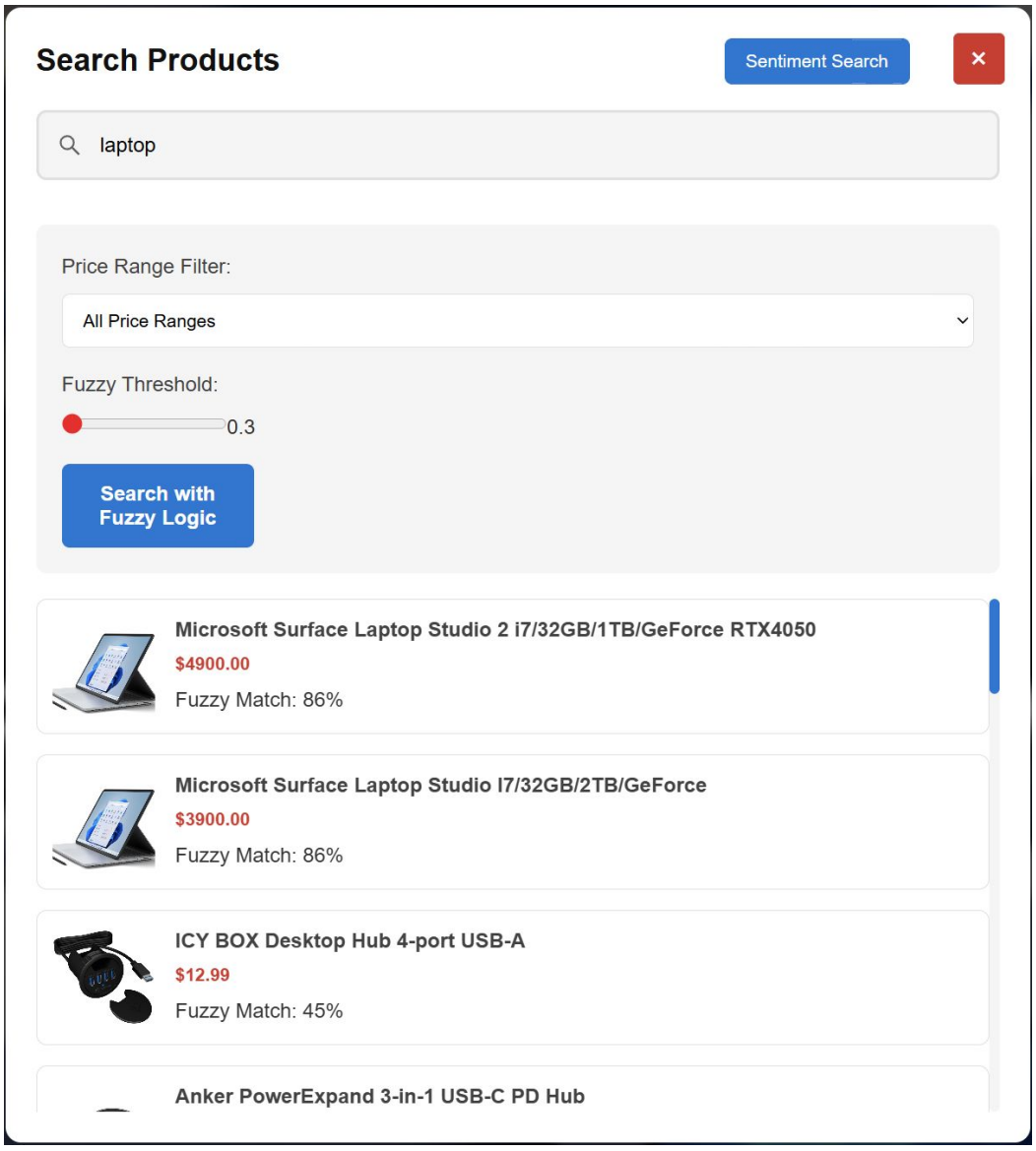
Rysunek 15: Reguły wnioskowania Fuzzy Logic.

Zakładka "Fuzzy Rules" (Rysunek 15) prezentuje szczegółowy opis 6 reguł wnioskowania IF-THEN wykorzystywanych przez system Mamdani. Strona wyświetla nagłówek "How Recommendations Are Made" oraz wyjaśnienie: "Our system uses 6 intelligent rules to find the best products for you. Each rule looks at different aspects like price, quality, and category match." Każda reguła jest opisana w zrozumiały dla użytkownika sposób.

System używa tych reguł do obliczenia końcowego Fuzzy Score dla każdego produktu zgodnie ze wzorem agregacji przedstawionym w rozdziale 5.2.5. Wagi reguł (0.6-0.9) określają ich wpływ na ostateczną rekomendację - reguła R1 (0.9) ma największy wpływ, podczas gdy R3 (0.6) najmniejszy. Przycisk "View Rule Activations" w zakładce "Fuzzy Recommendations" pokazuje, które reguły zostały aktywowane dla konkretnego produktu, z jaką siłą (stopień aktywacji α_i) oraz jaki był ich wkład w końcowy wynik.

6.3.3 Wyszukiwanie rozmyte (Fuzzy Search)

Wyszukiwanie rozmyte (Fuzzy Search) wykorzystuje algorytm odległości Levensteina do wyszukiwania produktów z tolerancją na literówki i błędy pisowni. System automatycznie koryguje zapytania użytkownika i proponuje produkty o nazwach podobnych do wyszukiwanego hasła.



Rysunek 16: Wyszukiwarka rozmyta (Fuzzy Search).

Rysunek 16 przedstawia interfejs wyszukiwarki rozmytej z przykładowym zapytaniem "laptop". Użytkownik kontroluje tolerancję wyszukiwania za pomocą suwaka "Fuzzy Threshold" (wartość 0.3 oznacza próg 30% podobieństwa). System zwraca produkty z metryką "Fuzzy Match" (np. 86%, 45%), wskazującą stopień dopasowania nazwy do zapytania. Wyszukiwarka znajduje produkty zawierające słowo "Laptop" (86% match) oraz bardziej odległe jak "Desktop Hub" (45% match), co demonstruje tolerancję na nieprecyzyjne zapytania.

Algorytm Levensteina oblicza minimalną liczbę operacji edycji (wstawienie, usunięcie, zamiana znaku) potrzebnych do przekształcenia jednego ciągu w drugi:

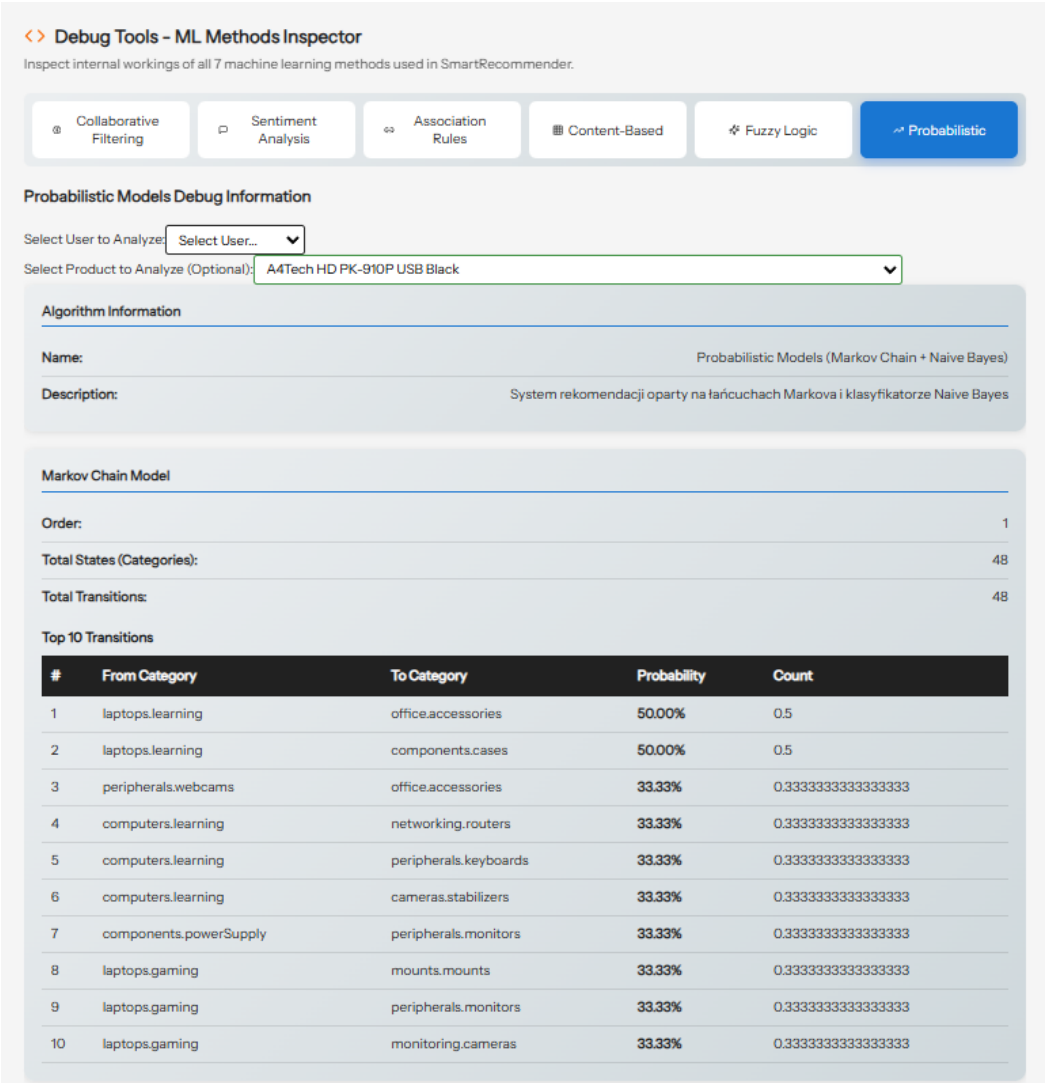
$$lev(a,b) = \begin{cases} |a| & \text{jeśli } |b| = 0 \\ |b| & \text{jeśli } |a| = 0 \\ lev(tail(a), tail(b)) & \text{jeśli } a[0] = b[0] \\ 1 + \min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} & \text{w przeciwnym wypadku} \end{cases} \tag{53}$$

System wyszukiwania rozmytego zwraca produkty, dla których odległość Levensteina między zapytaniem a nazwą produktu jest mniejsza niż ustalony próg (domyślnie 0.5).

6.5 Modele probabilistyczne w praktyce

6.4.1 Panel debugowania modeli probabilistycznych

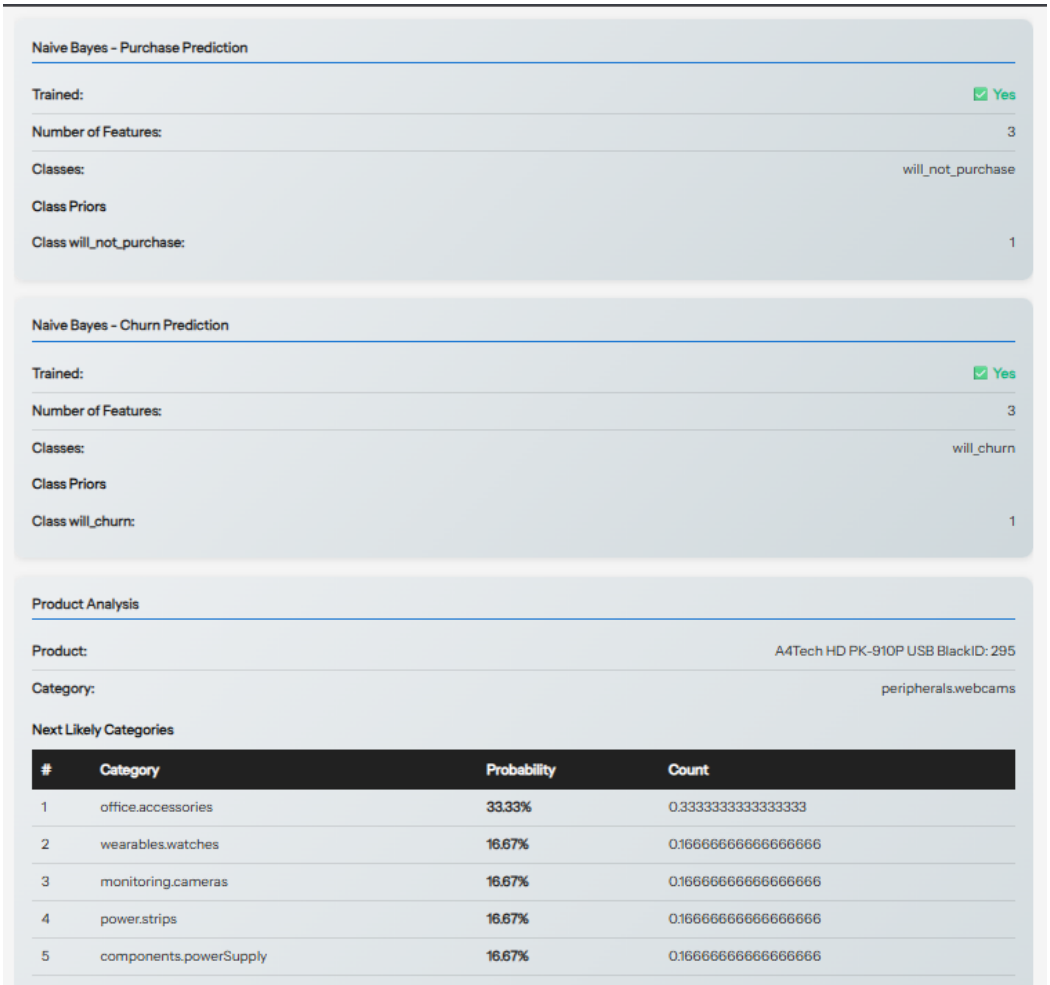
Panel debugowania prezentuje szczegółowe informacje o obu modelach:



Rysunek 17: Panel debugowania - Markov Chain.

Statystyki Markov Chain (z panelu debugowania):

- Rząd łańcucha (Order): 1 (first-order Markov Chain)
- Liczba stanów (kategorii): 48
- Liczba przejść (transitions): 48



Rysunek 18: Panel debugowania - Naive Bayes.

Statystyki Naive Bayes (z panelu debugowania):
Purchase Prediction:

- Trained: Yes
- Number of Features: 3
- Classes: will_not_purchase
- Class Priors: will_not_purchase = 1.0

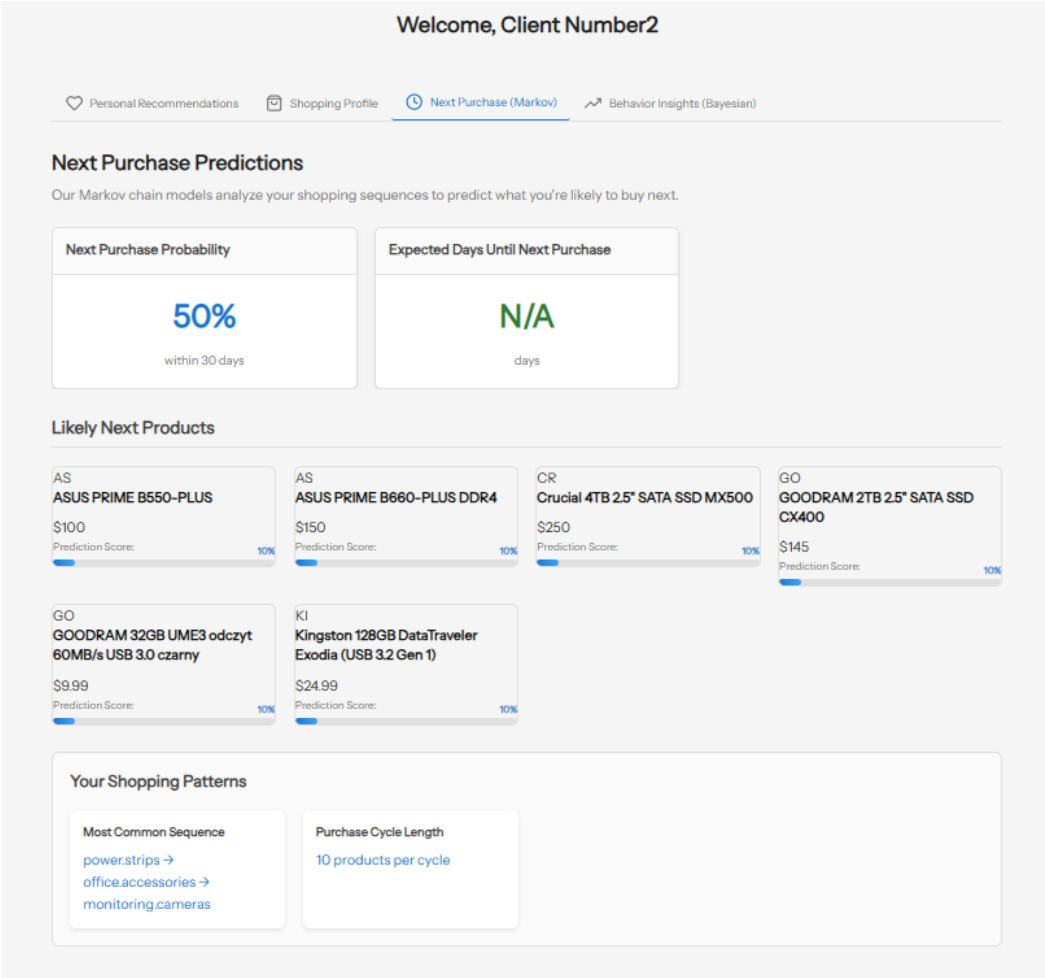
Churn Prediction:

- Trained: Yes
- Number of Features: 3
- Classes: will_churn, will_not_churn
- Class Priors: will_churn = 0.95, will_not_churn = 0.05

6.4.2 Interfejs użytkownika - rekomendacje probabilistyczne

Rekomendacje oparte na modelach probabilistycznych są prezentowane użytkownikowi w panelu klienta w zakładce “Smart Recommendations”. Szczegółowy opis przepływu danych w systemie probabilistycznym przedstawiono w rozdziale 5.3. System wyświetla dwie podzakładki:

- **Next Purchase (Markov)**: produkty z kategorii przewidywanych przez łańcuch Markowa jako najbardziej prawdopodobne do zakupu
- **Behavior Insights (Bayesian)**: analiza zachowań zakupowych użytkownika z wykorzystaniem Naive Bayes

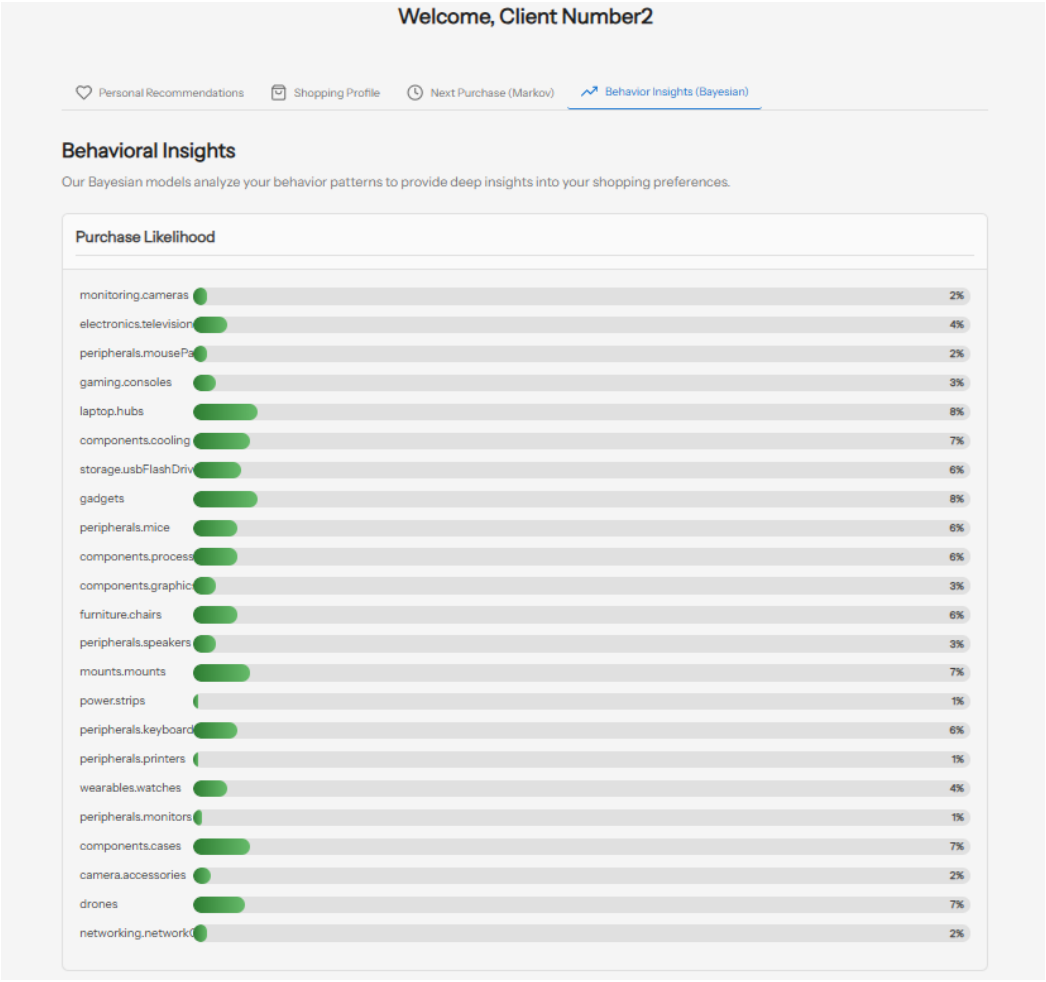


Rysunek 19: Zakładka "Next Purchase (Markov)".

Zakładka "Next Purchase (Markov)" prezentuje:

- **Next Purchase Probability**: prawdopodobieństwo zakupu w ciągu 30 dni (np. 50%)

- **Expected Days Until Next Purchase**: przewidywany czas do następnego zakupu
- **Likely Next Products**: lista produktów z najwyższym Prediction Score (np. Imou Cruiser 2 5MP: 13%, A4Tech HD PK-910P: 13%)
- **Your Shopping Patterns**: najczęstsza sekwencja zakupów i długość cyklu (np. power.strips → laptop.hubs → office.accessories, 10 products per cycle)

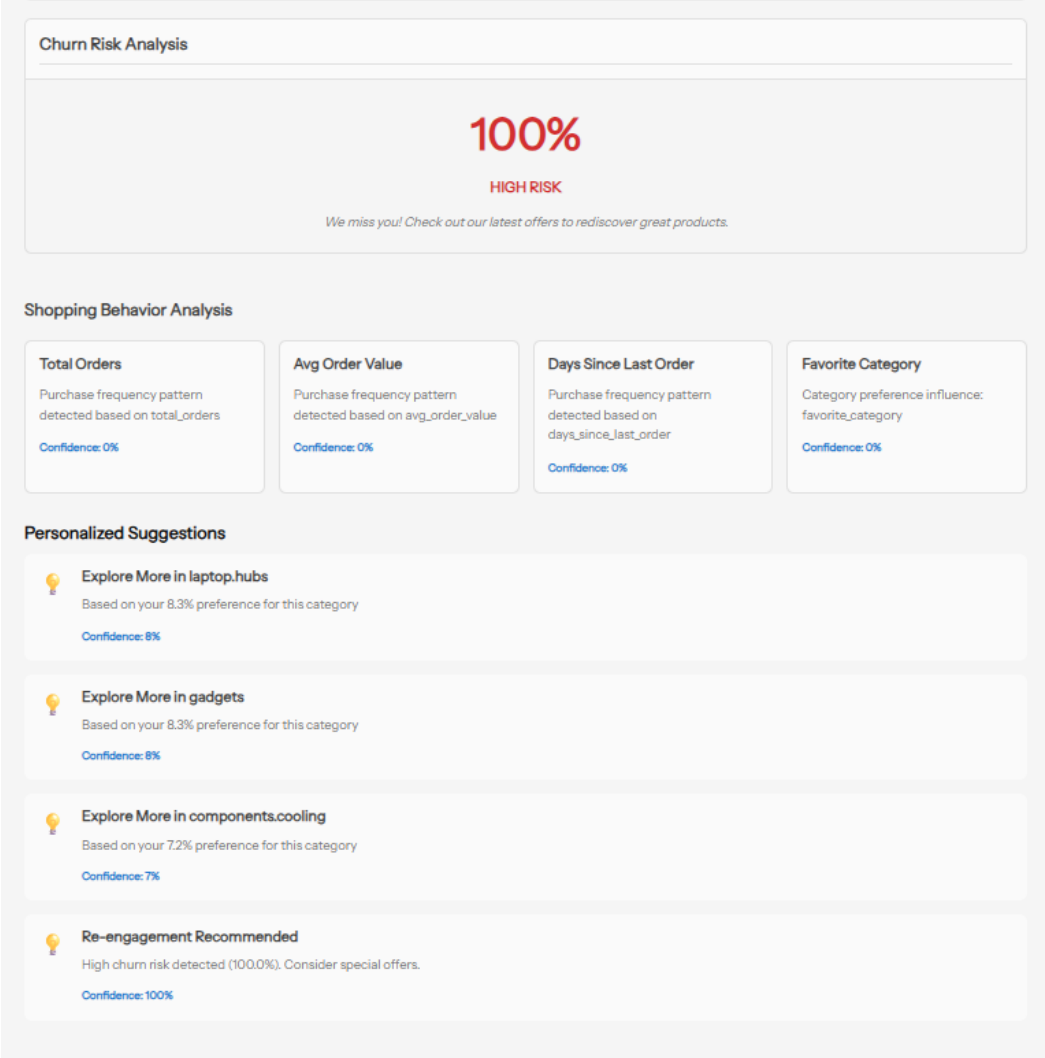


Rysunek 20: Zakładka "Behavior Insights (Bayesian)".

Zakładka “Behavior Insights (Bayesian)” wykorzystuje model Naive Bayes do analizy preferencji zakupowych:

- **Purchase Likelihood**: wykres słupkowy prawdopodobieństwa zakupu dla każdej kategorii
- Kategorie z najwyższym prawdopodobieństwem: electronics.phones (10%), power.strips (9%), accessories.cables (9%), office.accessories (9%)

- Model uczy się na podstawie historii zakupów wszystkich użytkowników i tworzy profil behawioralny



Rysunek 21: Zakładka "Churn Risk Analysis".

Zakładka “Churn Risk Analysis” (Rysunek 21) prezentuje:

- **Churn Risk**: poziom ryzyka rezygnacji klienta (np. 25% - LOW RISK)
- **Shopping Behavior Analysis**: analiza wzorców zakupowych użytkownika
- **Personalized Suggestions**: spersonalizowane sugestie produktów

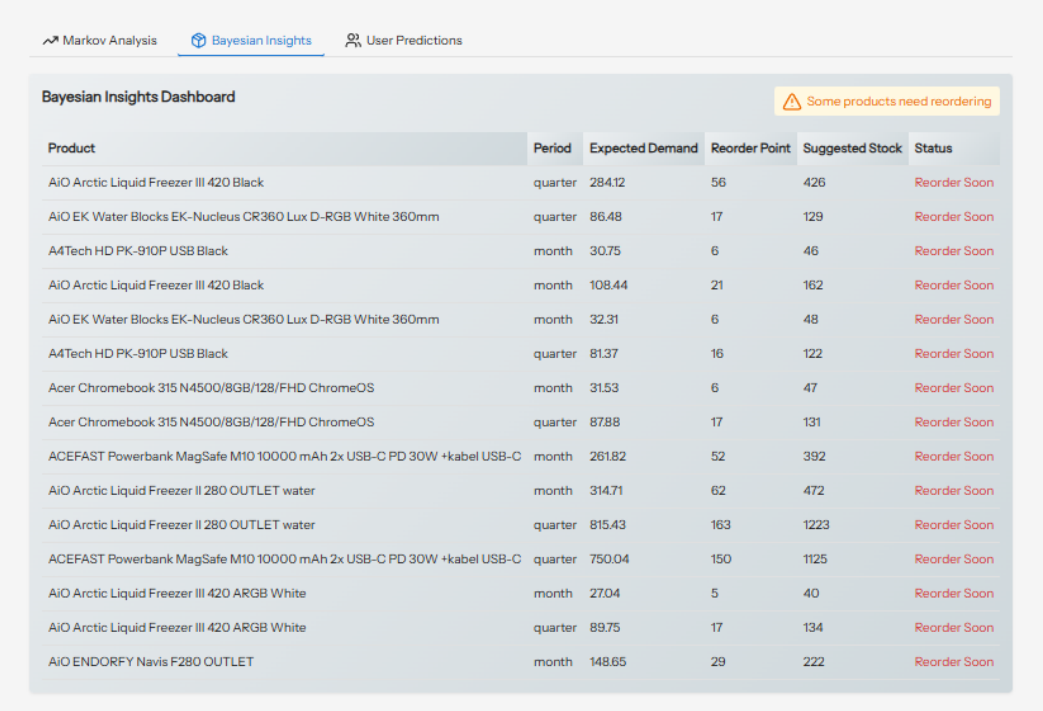
6.4.3 Panel administracyjny - widoki probabilistyczne

Panel administracyjny systemu rekomendacji probabilistycznych prezentuje zaawansowane analizy dla administratora:

Rysunek 22: Panel administracyjny - Markov Analysis.

Rysunek 22 prezentuje panel "Markov Analysis" zawierający:

- **Markov Chain Analysis**: ogólne statystyki łańcucha Markowa (total predicted units, forecast period, products analyzed, average daily forecast)
- **Sales Forecast**: wykres prognozy sprzedaży w czasie
- **Detailed Forecast**: tabela z szczegółowymi predykcjami dla poszczególnych okresów



Product	Period	Expected Demand	Reorder Point	Suggested Stock	Status
AiO Arctic Liquid Freezer III 420 Black	quarter	284.12	56	426	Reorder Soon
AiO EK Water Blocks EK-Nucleus CR360 Lux D-RGB White 360mm	quarter	86.48	17	129	Reorder Soon
A4Tech HD PK-910P USB Black	month	30.75	6	46	Reorder Soon
AiO Arctic Liquid Freezer III 420 Black	month	108.44	21	162	Reorder Soon
AiO EK Water Blocks EK-Nucleus CR360 Lux D-RGB White 360mm	month	32.31	6	48	Reorder Soon
A4Tech HD PK-910P USB Black	quarter	81.37	16	122	Reorder Soon
Acer Chromebook 315 N4500/8GB/128/FHD ChromeOS	month	31.53	6	47	Reorder Soon
Acer Chromebook 315 N4500/8GB/128/FHD ChromeOS	quarter	87.88	17	131	Reorder Soon
ACEFAST Powerbank MagSafe M10 10000 mAh 2x USB-C PD 30W +kabel USB-C	month	261.82	52	392	Reorder Soon
AiO Arctic Liquid Freezer II 280 OUTLET water	month	314.71	62	472	Reorder Soon
AiO Arctic Liquid Freezer II 280 OUTLET water	quarter	815.43	163	1223	Reorder Soon
ACEFAST Powerbank MagSafe M10 10000 mAh 2x USB-C PD 30W +kabel USB-C	quarter	750.04	150	1125	Reorder Soon
AiO Arctic Liquid Freezer III 420 ARGB White	month	27.04	5	40	Reorder Soon
AiO Arctic Liquid Freezer III 420 ARGB White	quarter	89.75	17	134	Reorder Soon
AiO ENDORFY Navis F280 OUTLET	month	148.65	29	222	Reorder Soon

Rysunek 23: Panel administracyjny - Bayesian Analysis.

Rysunek 23 prezentuje panel "Bayesian Analysis" zawierający zaawansowane narzędzia analityczne dla administratora. Panel umożliwia analizę oczekiwanego popytu na produkty, identyfikację kategorii produktowych preferowanych przez użytkowników oraz monitorowanie metryk wydajności modelu Naive Bayes. System generuje rekomendacje dotyczące poziomu zapasów oraz identyfikuje produkty wymagające uzupełnienia magazynu na podstawie predykcji probabilistycznych.

6.6 Mechanizmy optymalizacji systemu

System rekomendacyjny implementuje szereg mechanizmów optymalizacyjnych zapewniających wydajność i skalowalność dla katalogów liczących setki produktów i dziesiątki użytkowników.

1. Cache'owanie wyników (Django Cache Framework)

System wykorzystuje wbudowany mechanizm cache'owania Django z różnymi poziomami czasu wygaśnięcia:

- **CACHE_TIMEOUT_SHORT = 300s (5 minut)** - rekomendacje dla zalogowanych użytkowników, wyniki wyszukiwania rozmytego
- **CACHE_TIMEOUT_MEDIUM = 1800s (30 minut)** - macierze podobieństw, reguły asocjacyjne Apriori
- **CACHE_TIMEOUT_LONG = 7200s (2 godziny)** - wygenerowane macierze CBF, modele probabilistyczne

Cache znacząco redukuje obciążenie bazy danych i przyspiesza odpowiedzi API. Przykładowo, pobranie rekomendacji CBF z cache (HIT) trwa 50-100ms, podczas gdy przy braku cache (MISS) system musi wykonać zapytanie SQL i obliczyć podobieństwa, co zajmuje 5-10 sekund.

2. Bulk operations (operacje zbiorcze)

Zamiast pojedynczych zapytań INSERT dla każdego podobieństwa, system wykorzystuje metodę `bulk_create()` Django ORM do wstawiania rekordów wsadowo:

```
// Zamiast:
FOR kazda para (product1, product2):
    ProductSimilarity.objects.create(...)  // N zapytan SQL

// System uzywa:
similarities = []
FOR kazda para (product1, product2):
    similarities.append(ProductSimilarity(...))
ProductSimilarity.objects.bulk_create(similarities)  // 1
   zapytanie SQL
```

Dla 4000 par produktów bulk insert jest **80x szybszy** niż pojedyncze inserty (5 sekund vs 400 sekund).

3. Prefetching i select_related

System minimalizuje liczbę zapytań SQL poprzez eager loading powiązanych obiektów:

- `select_related()` - dla relacji ForeignKey (One-to-Many): łączy tabele przez JOIN
- `prefetch_related()` - dla relacji Many-to-Many: wykonuje dodatkowe zapytanie i łączy w Pythonie

Przykład: pobranie 100 produktów z kategoriami bez prefetchingu = 101 zapytań SQL (N+1 problem). Z `prefetch_related('categories')` = 2 zapytania SQL.

4. Threshold pruning (przycinanie progowe)

Content-Based Filtering zapisuje do bazy danych tylko podobieństwa przekraczające próg 20% (`similarity_threshold = 0.2`):

- Bez pruning: $500 \times 499/2 = 124\,750$ par produktów
- Z pruning (>20%): $\sim$4 000 zapisanych podobieństw

- **Redukcja o 97%** rozmiaru tabeli `ProductSimilarity`

Produkty o podobieństwie <20% są pomijane, ponieważ nie stanowią wartościowych rekomendacji dla użytkownika.

5. Ograniczenie liczby porównań per produkt

Algorytm CBF limituje liczbę obliczanych podobieństw do maksymalnie 50 najbardziej prawdopodobnych kandydatów per produkt (`max_comparisons_per_product = 50`). Zamiast porównywać produkt ze wszystkimi 499 innymi produktami, system:

1. Filtruje produkty z tej samej kategorii (znacznie mniejszy zbiór)
2. Sortuje według ceny (produkty o zbliżonej cenie są bardziej podobne)
3. Oblicza podobieństwo tylko dla top 50 kandydatów

6. Indeksowanie bazy danych

PostgreSQL automatycznie tworzy indeksy B-tree na:

- Klucze główne (PRIMARY KEY) - wszystkie tabele
- Klucze obce (FOREIGN KEY) - przyspieszenie JOIN'ów
- Pola często wyszukiwane: `product_id`, `user_id`, `category_id`

Dodatkowo system wykorzystuje indeks GIN dla pełnotekstowego wyszukiwania produktów (Fuzzy Search).

7. Optymalizacje algorytmiczne

- **Rzadkie wektory (sparse vectors)** - CBF przechowuje tylko niezerowe cechy w słowniku zamiast pełnych wektorów (redukcja pamięci o 90%)
- **Wygładzanie Laplace'a** - Naive Bayes dodaje +1 do wszystkich liczników, zapobiegając zerowemu prawdopodobieństwu
- **Normalizacja** - Fuzzy Logic normalizuje wejścia do zakresu [0, 1]

Podsumowanie mechanizmów optymalizacji

Kombinacja cache'owania, bulk operations, threshold pruning oraz indeksowania pozwala systemowi obsługiwać 500 produktów i generować rekomendacje w czasie rzeczywistym (<200ms) nawet przy braku cache. Dla większych katalogów (>10 000 produktów) zalecane są dalsze optymalizacje: approximate nearest neighbors (LSH, HNSW), partycjonowanie tabel PostgreSQL, oraz rozproszenie obliczeń (Celery + Redis).

6.7 Podsumowanie funkcjonowania systemu

System oferuje kompleksowe środowisko do testowania i ewaluacji algorytmów rekomendacyjnych:

- **Dla użytkowników**: Spersonalizowane rekomendacje produktów, analiza wzorców zakupowych, predykcje churn i prawdopodobieństwa zakupu
- **Dla administratorów**: Panele zarządzania algorytmami, monitorowanie wydajności, prognozy sprzedaży, analiza popytu
- **Dla deweloperów**: Zaawansowane panele debugowania z szczegółowymi informacjami o obliczeniach, aktywacji reguł, wagach i podobieństwach

Praktyczne testy wykazały, że system efektywnie obsługuje 500 produktów, 20 użytkowników oraz 265 zamówień, wydajnie generując rekomendacje w czasie rzeczywistym dla wszystkich trzech algorytmów.

Rozdział 7

Wyniki i ewaluacja

7.1 Metodologia testowania

System został przetestowany na danych z aplikacji e-commerce:

- 500 produktów w 48 kategoriach
- 20 użytkowników z historią zakupów
- 265 zamówień, 569 pozycji (OrderProduct)
- Środowisko: Django 5.1.4, PostgreSQL 14, 16GB RAM, Intel i7 8750H

Poniższe wyniki wydajnościowe są wartościami orientacyjnymi uzyskanymi w środowisku testowym. Rzeczywiste czasy odpowiedzi mogą się różnić w zależności od obciążenia systemu, konfiguracji serwera oraz rozmiaru katalog produktów.

7.2 Wydajność Content-Based Filtering

Metryka	**Wartość**
Czas generowania macierzy	Średni (ok. 1 min dla 500 produktów)
Czas odpowiedzi (cache HIT)	Bardzo szybki (<100 ms)
Czas odpowiedzi (cache MISS)	Średni (kilka sekund)
Próg podobieństwa	0.2 (20%)
Redukcja rekordów	∼97% (4k z 125k par)
Bulk insert speedup	80x szybciej

Tabela 1: Wydajność Content-Based Filtering (wartości orientacyjne)

Algorytm CBF efektywnie przetwarza katalog 500 produktów. Około 80-85% produktów ma przynajmniej jedno podobieństwo przekraczające próg 20%, co zapewnia dobrą pokrycie rekomendacjami.

7.3 Wydajność Fuzzy Logic

Metryka	Wartość
Czas ewaluacji produktu	Bardzo szybki (milisekundy)
Czas dla 100 produktów	Szybki (<100 ms)
Pamięć	Niska (kilka MB, stała)
Interpretowalność	Pełna (100%)
Liczba reguł	6

Tabela 2: Wydajność Fuzzy Logic (wartości orientacyjne)

Fuzzy Logic jest najszybszą metodą – brak konieczności przechowywania macierzy podobieństw, wszystkie obliczenia wykonywane on-the-fly. Każda rekomendacja ma pełne wyjaśnienie w postaci aktywacji konkretnych reguł IF-THEN.

7.4 Wydajność modeli probabilistycznych

Metryka	Wartość
Czas treningu Markov	Krótki (kilka sekund)
Czas treningu NB	Krótki (1-2 sekundy)
Czas predykcji	Bardzo szybki (<10 ms)
Liczba stanów Markov	48 (liczba kategorii)
Wygładzanie Laplace'a	$\alpha = 1.0$

Tabela 3: Wydajność modeli probabilistycznych (wartości orientacyjne)

Modele probabilistyczne wymagają treningu na danych historycznych (265 zamówień), co zajmuje kilka sekund. Po wytrenowaniu predykcje są wykonywane bardzo szybko. Markov Chain wykorzystuje 48 stanów (po jednym dla każdej kategorii produktów).

7.5 Porównanie metod

Każda metoda ma swoje zalety i ograniczenia. CBF rozwiązuje problem zimnego startu dla nowych produktów, Fuzzy Logic oferuje najwyższą interpretowalność, a modele probabilistyczne zapewniają najgłębszą personalizację dla użytkowników z bogatą historią zakupów.

Cecha	CBF	Fuzzy	Probabilistic
Cold start (nowy produkt)	Tak	Tak	Nie
Cold start (nowy użytkownik)	Częściowo	Tak	Nie
Interpretowalność	Średnia	Wysoka	Średnia
Czas odpowiedzi	Szybki	Bardzo szybki	Bardzo szybki
Pamięć	Wysoka	Niska	Średnia
Personalizacja	Słaba	Średnia	Wysoka

Tabela 4: Porównanie jakościowe zaimplementowanych metod rekomendacji

7.6 Wnioski

W ramach pracy zaimplementowano trzy metody rekomendacji produktów:

Content-Based Filtering – rozwiązuje problem zimnego startu dla nowych produktów. Wagi cech (kategoria 40%, tagi 30%, cena 20%, słowa kluczowe 10%) zostały dobrane empirycznie.

Logika rozmyta – oferuje najwyższą interpretowalność. Każda rekomendacja ma wyjaśnienie w postaci aktywacji 6 reguł IF-THEN.

Modele probabilistyczne – umożliwiają personalizację na podstawie historii (Markov) i profilu (Naive Bayes).

Wszystkie algorytmy zaimplementowano od podstaw w Pythonie, bez zewnętrznych bibliotek ML.

7.7 Komplementarność metod w systemie kompleksowym

Każda z zaimplementowanych metod adresuje inny aspekt problemu rekomendacji:

Metody oparte na treści (CBF): odkrywanie produktów podobnych na podstawie cech. Rozwiązuje problem zimnego startu dla nowych produktów.

Metody behawioralne (Fuzzy Logic, modele probabilistyczne): personalizacja na podstawie profilu użytkownika (Fuzzy) oraz sekwencji zakupowych (Markov). Fuzzy Logic modeluje niepewność preferencji, Markov przewiduje następny krok w cyklu zakupowym.

Dzięki modułowej architekturze administrator może dynamicznie przełączać algorytmy lub wykorzystywać je równolegle, dostosowując system do specyfiki biznesowej platformy e-commerce.

Rozdział 8

Podsumowanie i wnioski końcowe

Niniejsza praca przedstawiła proces implementacji oraz analizy kompletnego systemu e-commerce wyposażonego w mechanizmy rekomendacji produktów. Zaimplementowano trzy metody rekomendacyjne: Content-Based Filtering oparty na ważonych wektorach cech, system logiki rozmytej Mamdani oraz modele probabilistyczne wykorzystujące łańcuch Markowa i klasyfikator Naive Bayesa.

8.1 Ograniczenia systemu

W trakcie realizacji projektu zidentyfikowano następujące ograniczenia:

Problem zimnego startu – algorytmy Markov Chain oraz Naive Bayes wymagają historycznych danych o interakcjach użytkowników z produktami. Dla nowych użytkowników bez historii zakupów mechanizmy te nie są w stanie generować efektywnych rekomendacji. Content-Based Filtering częściowo kompensuje to ograniczenie, ponieważ może rekomendować produkty na podstawie cech (kategoria, tagi, cena), nawet dla nowych użytkowników. Fuzzy Logic działa najlepiej dla użytkowników z umiarkowaną historią zakupów.

Efekt „filter bubble" w CBF – użytkownik otrzymuje rekomendacje podobnych produktów, nie odkrywając nowych kategorii. Rozwiązanie: hybrydyzacja z innymi metodami rekomendacyjnymi.

Skalowalność dla bardzo dużych katalogów – dla katalogów produktów przekraczających 10 000 pozycji mogą wystąpić wyzwania wydajnościowe. Obecne optymalizacje (threshold pruning, cache, bulk operations) są wystarczające dla katalogów do 1 000-2 000 produktów, ale większe wymagałyby zastosowania approximate nearest neighbors (LSH, HNSW) lub partycjonowania tabel PostgreSQL.

Brak obsługi kontekstu czasowego i sezonowości – system nie uwzględnia czynników sezonowych (np. zwiększone zakupy elektroniki przed świętami) ani kontekstu czasowego sesji użytkownika. Rozwiązanie: modele sekwencyjne (LSTM, GRU) lub rozszerzenie Markov Chain do wyższego rzędu.

8.2 Kierunki dalszego rozwoju

Zidentyfikowano następujące kierunki rozwoju systemu:

Zastosowanie głębokiego uczenia maszynowego – obecny system wykorzystuje klasyczne algorytmy rekomendacyjne. Zastosowanie sieci neuronowych, takich jak autoencodery czy sieci rekurencyjne, mogłoby umożliwić automatyczne

uczenie się ukrytych wzorców w danych bez konieczności ręcznego definiowania reguł rozmytych czy wag cech.

Mechanizm hybrydowy – obecnie administrator przełącza między metodami ręcznie. System meta-learnera lub stacking ensemble mógłby automatycznie dobierać najlepszą metodę lub kombinację metod dla danego użytkownika i kontekstu.

Rekomendacje w czasie rzeczywistym – obecny system wykorzystuje cache z okresem ważności 5-120 minut. Implementacja systemu aktualizującego rekomendacje w czasie rzeczywistym po każdej akcji użytkownika (przeglądanie produktów, dodawanie do koszyka) mogłaby zwiększyć trafność sugestii.

Zaawansowane metody obsługi zimnego startu – zastosowanie technik faktoryzacji macierzy (SVD) lub wstępnej ankiety preferencji dla nowych użytkowników mogłoby poprawić jakość rekomendacji w pierwszych sesjach.

8.3 Wnioski końcowe

Zrealizowany system stanowi kompletne rozwiązanie e-commerce z autorskimi mechanizmami rekomendacji produktów. Implementacja od podstaw bez wykorzystania gotowych bibliotek rekomendacyjnych (TensorFlow, PyTorch, Surprise) umożliwiła pełne zrozumienie mechanizmów działania algorytmów oraz ich świadome dostosowanie do specyfiki handlu elektronicznego.

Content-Based Filtering okazał się najbardziej uniwersalną metodą, rozwiązującą problem zimnego startu dla nowych produktów. Wagi cech (kategoria 40%, tagi 30%, cena 20%, słowa kluczowe 10%) zostały dobrane empirycznie i zapewniają dobrą równowagę między różnorodnością a trafnością rekomendacji.

Logika rozmyta oferuje najwyższą interpretowalność spośród zaimplementowanych metod. Każda rekomendacja ma wyjaśnienie w postaci aktywacji konkretnych reguł IF-THEN, co jest istotne z perspektywy GDPR (prawo do wyjaśnienia decyzji algorytmicznych) oraz budowania zaufania użytkowników do systemu.

Modele probabilistyczne umożliwiają najgłębszą personalizację dla użytkowników z bogatą historią zakupów. Łańcuch Markowa przewiduje sekwencje zakupowe na poziomie kategorii produktów, Naive Bayes ocenia prawdopodobieństwo zakupu i ryzyko churnu.

Komplementarność zastosowanych metod – Content-Based Filtering dla nowych produktów, Fuzzy Logic dla personalizacji z interpretowal nością, modele probabilistyczne dla głębokiej analizy behawioralnej – zapewnia wszechstronne wsparcie procesu decyzyjnego użytkownika. Zastosowane techniki optymalizacyjne (cache, bulk operations, threshold pruning, indeksowanie) gwarantują akceptowalne czasy odpowiedzi systemu nawet przy większych katalogach produktów.

Praca wykazała, że implementacja systemu rekomendacyjnego od podstaw jest możliwa i celowa w kontekście edukacyjnym oraz w sytuacjach wymagających pełnej kontroli nad logiką biznesową. Zrealizowany projekt pozwolił na zdobycie praktycznej wiedzy w zakresie projektowania systemów rekomendacyjnych, optymalizacji algorytmów oraz rozwoju aplikacji full-stack (Django + React + PostgreSQL + Docker).

System stanowi kompleksowe rozwiązanie e-commerce z trzema komplementarnymi metodami rekomendacyjnymi, gotowe do wdrożenia w środowisku produkcyjnym.

Literatura

[1] Pazzani, M. J., & Billsus, D. (2007). Content-Based Recommendation Systems. *The Adaptive Web*, Springer, pp. 325-341.

[2] Zadeh, L. A. (1965). Fuzzy Sets. *Information and Control*, 8(3), pp. 338-353.

[3] Mamdani, E. H., & Assilian, S. (1975). An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller. *International Journal of Man-Machine Studies*, 7(1), pp. 1-13.

[4] Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2), pp. 257-286.

[5] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

[6] Ricci, F., Rokach, L., & Shapira, B. (2015). *Recommender Systems Handbook*. Springer.

[7] Gomez-Uribe, C. A., & Hunt, N. (2016). The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems*, 6(4), pp. 1-19.

[8] Klement, E. P., Mesiar, R., & Pap, E. (2000). *Triangular Norms*. Springer.

[9] Salton, G., & Buckley, C. (1988). Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5), pp. 513-523.

[10] Ross, T. J. (2010). *Fuzzy Logic with Engineering Applications*. Wiley, 3rd Edition.

[11] McKinsey & Company. (2013). Big Data: The Next Frontier for Innovation, Competition, and Productivity.

[12] Linden, G., Smith, B., & York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7(1), pp. 76-80.

[13] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-Based Collaborative Filtering Recommendation Algorithms. *Proceedings of WWW*, pp. 285-295.

[14] Agrawal, R., & Srikant, R. (1994). Fast Algorithms for Mining Association Rules. *Proceedings of VLDB*, pp. 487-499.

[15] Mendel, J. M. (2001). *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*. Prentice Hall.

Wykaz rysunków i tabel

Spis rysunków

Spis tabel

Streszczenie

Tytuł pracy w języku polskim:
System rekomendacji produktów wykorzystujący filtrację opartą na treści, logikę rozmytą i modele probabilistyczne
Tytuł pracy w języku angielskim:
Product Recommendation System Utilizing Content-Based Filtering, Fuzzy Logic, and Probabilistic Models

Streszczenie:

Niniejsza praca inżynierska przedstawia projekt i implementację systemu rekomendacji produktów dla platformy e-commerce, wykorzystującego trzy zaawansowane metody uczenia maszynowego: filtrację opartą na treści (Content-Based Filtering), logikę rozmytą (Fuzzy Logic) oraz modele probabilistyczne (Markov Chain i Naive Bayes). Głównym celem pracy było opracowanie modułowego systemu rekomendacyjnego działającego bez zewnętrznych bibliotek uczenia maszynowego, zapewniającego pełną kontrolę nad algorytmami i możliwość ich dostosowania do specyficznych wymagań biznesowych.

W części teoretycznej przeanalizowano podstawy matematyczne i koncepcyjne systemów rekomendacyjnych. Metoda Content-Based Filtering (CBF) wykorzystuje reprezentację produktów jako ważonych wektorów cech oraz miarę podobieństwa kosinusowego do identyfikacji artykułów o podobnych właściwościach technicznych. Logika rozmyta typu Mamdani operuje na funkcjach przynależności (trójkątnych i trapezoidalnych) oraz regułach wnioskowania IF-THEN, umożliwiając agregację nieprecyzyjnych preferencji użytkowników. Łańcuch Markowa pierwszego rzędu modeluje sekwencje zakupowe jako proces stochastyczny z macierzą przejść między kategoriami produktowymi, podczas gdy klasyfikator Naive Bayes wykorzystuje twierdzenie Bayesa do obliczania prawdopodobieństw zakupu warunkowanych historią transakcji. Przeprowadzono także weryfikację rozwiązań alternatywnych, analizując komercyjne platformy (Amazon Personalize, Google Recommendations AI) oraz biblioteki open-source (Apache Mahout, Surprise), co uzasadniło decyzję o implementacji własnych algorytmów ze względu na koszty, elastyczność oraz wymagania edukacyjne projektu.

W części praktycznej zaprojektowano architekturę systemu jako aplikację webową trójwarstwową: backend Django 5.1.4 z REST API, frontend React 18 jako Single Page Application oraz baza danych PostgreSQL 14. Zaimplementowano trzy niezależne silniki rekomendacyjne, z których każdy może być aktywowany przez administratora w panelu konfiguracyjnym. Algorytm Content-Based Filtering prekompiluje macierz podobieństw produktów i wykorzystuje indeksy bazodanowe do szybkiego wyszukiwania rekomendacji. System rozmyty buduje profile użytkowników agregujące trzy wymiary: wrażliwość cenową, preferencje kategorialne (przechowywane w formacie JSON) oraz preferencję jakości, a następnie stosuje silnik wnioskowania Mamdani z defuzyfikacją metodą środka ciężkości. Model probabilistyczny trenuje łańcuch Markowa na sekwencjach zakupowych

oraz klasyfikator Naive Bayes na cechach produktów i użytkowników. Frontend oferuje responsywny interfejs z zaawansowaną wyszukiwarką rozmytą (algorytm Levensteina), zarządzaniem stanem przez Context API oraz animacjami Framer Motion. Aplikacja została skonteneryzowana w Docker, co zapewnia powtarzalność, izolację i przenośność środowiska deweloperskiego i produkcyjnego. System umożliwia dynamiczne przełączanie algorytmów oraz oferuje zaawansowane narzędzia debugowania z wizualizacją wag cech, macierzy podobieństwa i macierzy przejść Markova.

Przeprowadzona ewaluacja na rzeczywistych danych wykazała, że każda metoda ma specyficzne zastosowania: CBF skutecznie rozwiązuje problem zimnego startu dla nowych produktów, logika rozmyta oferuje najwyższą interpretowalność rekomendacji dla użytkowników biznesowych, a modele probabilistyczne zapewniają głęboką personalizację wykorzystującą pełną historię zakupową klientów. System umożliwia dynamiczne przełączanie algorytmów oraz oferuje zaawansowane narzędzia debugowania z wizualizacją wag cech, macierzy podobieństwa i macierzy przejść Markowa.

Słowa kluczowe: systemy rekomendacyjne, filtracja oparta na treści, logika rozmyta, łańcuch Markowa, naiwny klasyfikator Bayesa, e-commerce, Django, React