# Sistem preporuka u aplikaciji Flora

## Opis sistema preporuka

Pametni sistem preporuka implementiran u aplikaciji **Flora** temelji se na **item-based pristupu**. Sistem koristi podatke o prethodnim korisničkim narudžbama i analizira povezanost između proizvoda pomoću **ML.NET Matrix Factorization algoritma**.

* Kada korisnik kupi određeni proizvod (npr. buket ruža), sistem računa sličnost tog proizvoda sa drugim proizvodima koje su kupovali i drugi korisnici.
* Sistem predlaže **Top N proizvoda** s najvećim sličnostima (npr. aranžmane ili bukete sličnog tipa).
* Ako korisnik nema narudžbi, sistem vraća **featured ili new proizvode**.
* Algoritam se **dinamički uči** – kako raste broj narudžbi, preporuke postaju preciznije.

Time aplikacija korisnicima olakšava izbor i povećava angažman i zadovoljstvo.

## Putanja i kod glavne logike servisa sistema preporuka

**Putanja:** FloraApp\_RS2\Flora backend\Flora.Services\Services\RecommendationService

public async Task<List<ProductResponse>> GetRecommendedProductsAsync(int productId, int topN = 5)

{

try

{

if (\_similarityMap.Count == 0)

{

await RecalculateSimilarityMapAsync();

}

var recommendedProductIds = \_similarityMap

.Where(kv => kv.Key.Item1 == productId)

.OrderByDescending(kv => kv.Value)

.Take(topN)

.Select(kv => kv.Key.Item2)

.ToList();

using var context = \_contextFactory.CreateDbContext();

var products = await context.Products

.Where(p => recommendedProductIds.Contains(p.Id) && p.IsAvailable && p.Active)

.Include(p => p.Images)

.Include(p => p.Category)

.Include(p => p.Occasion)

.ToListAsync();

var result = recommendedProductIds

.Select(id => products.FirstOrDefault(p => p.Id == id))

.Where(p => p != null)

.Select(p => new ProductResponse

{

Id = p.Id,

Name = p.Name,

Description = p.Description,

Price = p.Price,

IsNew = p.IsNew,

IsFeatured = p.IsFeatured,

CategoryId = p.CategoryId,

CategoryName = p.Category?.Name,

OccasionId = p.OccasionId,

OccasionName = p.Occasion?.Name,

Active = p.Active,

IsAvailable = p.IsAvailable,

ImageUrls = p.Images.Select(i => i.ImageUrl).ToList()

})

.ToList();

return result;

}

catch (Exception ex)

{

\_logger.LogError(ex, "Greška prilikom dohvaćanja preporučenih proizvoda za proizvod ID: {ProductId}", productId);

return new List<ProductResponse>();

}

}

public async Task<List<ProductCoPurchase>> GetCoPurchaseMapAsync()

{

try

{

using var context = \_contextFactory.CreateDbContext();

var coPurchases = await context.OrderDetails

.Include(od => od.Order)

.AsNoTracking()

.GroupBy(od => od.OrderId)

.Select(orderGroup => new

{

OrderId = orderGroup.Key,

Products = orderGroup.Select(od => od.ProductId).Where(id => id.HasValue).Select(id => id.Value).ToList()

})

.ToListAsync();

var productPairs = new List<(int, int)>();

foreach (var order in coPurchases)

{

for (int i = 0; i < order.Products.Count; i++)

{

for (int j = 0; j < order.Products.Count; j++)

{

if (i != j)

{

productPairs.Add((order.Products[i], order.Products[j]));

}

}

}

}

var result = productPairs

.GroupBy(pair => pair)

.Select(g => new ProductCoPurchase

{

ProductId = g.Key.Item1,

CoPurchasedProductId = g.Key.Item2,

Count = g.Count()

})

.ToList();

return result;

}

catch (Exception ex)

{

\_logger.LogError(ex, "Greška prilikom generiranja mape ko-kupovina");

return new List<ProductCoPurchase>();

}

}

public async Task RecalculateSimilarityMapAsync()

{

try

{

\_logger.LogInformation("Započinje treniranje ML.NET item-based preporuka");

// Prvo dohvatimo sve potrebne podatke iz baze u memoriju

List<RecommendationInput> userProductPurchases;

// Koristimo poseban scope za dohvaćanje podataka iz baze

using (var context = \_contextFactory.CreateDbContext())

{

userProductPurchases = await context.OrderDetails

.Include(od => od.Order)

.Where(od => od.ProductId.HasValue)

.AsNoTracking()

.Select(od => new RecommendationInput

{

userId = od.Order.UserId.ToString(),

productId = od.ProductId.Value.ToString(),

Label = 1f

})

.Distinct()

.ToListAsync();

}

var uniqueUsers = userProductPurchases.Select(p => p.userId).Distinct().Count();

var uniqueProducts = userProductPurchases.Select(p => p.productId).Distinct().Count();

var totalPurchases = userProductPurchases.Count;

\_logger.LogInformation("Podaci za treniranje: {TotalPurchases} kupovina, {UniqueUsers} korisnika, {UniqueProducts} proizvoda",

totalPurchases, uniqueUsers, uniqueProducts);

if (uniqueUsers < 2 || uniqueProducts < 2 || totalPurchases < 10)

{

\_logger.LogWarning("Premalo podataka za treniranje MF modela. Potrebno je barem 2 korisnika, 2 proizvoda i 10 kupovina.");

\_similarityMap.Clear();

return;

}

var density = (double)totalPurchases / (uniqueUsers \* uniqueProducts);

if (density < 0.01) // Manje od 1% popunjenosti matrice

{

\_logger.LogWarning("Premala gustoća podataka za treniranje MF modela: {Density:P2}. Koristit ćemo jednostavniju metodu.", density);

}

var mlContext = new MLContext();

var dataView = mlContext.Data.LoadFromEnumerable(userProductPurchases);

var pipeline = mlContext.Transforms.Conversion

.MapValueToKey(inputColumnName: nameof(RecommendationInput.userId),

outputColumnName: "userIdEncoded")

.Append(mlContext.Transforms.Conversion

.MapValueToKey(inputColumnName: nameof(RecommendationInput.productId),

outputColumnName: "productIdEncoded"));

var transformedData = pipeline.Fit(dataView).Transform(dataView);

var options = new MatrixFactorizationTrainer.Options

{

MatrixColumnIndexColumnName = "userIdEncoded",

MatrixRowIndexColumnName = "productIdEncoded",

LabelColumnName = nameof(RecommendationInput.Label),

NumberOfIterations = 5,

ApproximationRank = 8,

Lambda = 0.1,

LearningRate = 0.01,

Quiet = true

};

// Treniramo model

var trainer = mlContext.Recommendation().Trainers.MatrixFactorization(options);

var model = trainer.Fit(transformedData);

var completeModel = pipeline.Append(trainer);

var trainedModel = completeModel.Fit(dataView);

var predictionEngine = mlContext.Model.CreatePredictionEngine<RecommendationInput, RecommendationPrediction>(trainedModel);

var newSimilarityMap = new Dictionary<(int, int), double>();

var productIds = userProductPurchases.Select(p => int.Parse(p.productId)).Distinct().ToList();

try {

foreach (var p1 in productIds)

{

foreach (var p2 in productIds)

{

if (p1 == p2) continue;

try {

var prediction = predictionEngine.Predict(new RecommendationInput

{

userId = p1.ToString(),

productId = p2.ToString()

});

// Dodajemo fallback za NaN ili Inf vrijednosti

double score = double.IsNaN(prediction.Score) || double.IsInfinity(prediction.Score)

? 0.0

: prediction.Score;

score = Math.Max(-5.0, Math.Min(5.0, score));

newSimilarityMap[(p1, p2)] = score;

}

catch (Exception ex) {

\_logger.LogWarning("Greška prilikom predviđanja sličnosti za proizvode {P1} i {P2}: {Message}",

p1, p2, ex.Message);

newSimilarityMap[(p1, p2)] = 0.0;

}

}

}

}

catch (Exception ex) {

\_logger.LogError(ex, "Greška prilikom izračunavanja sličnosti proizvoda");

newSimilarityMap = new Dictionary<(int, int), double>();

}

\_similarityMap = newSimilarityMap;

\_logger.LogInformation("ML.NET sličnosti proizvoda izračunate. Ukupno parova: {Count}", \_similarityMap.Count);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Greška prilikom izračunavanja ML.NET sličnosti proizvoda");

}

}

public async Task<List<ProductResponse>> GetRecommendedForUserAsync(int userId, int maxResults = 10)

{

try

{

using var context = \_contextFactory.CreateDbContext();

\_logger.LogInformation("Dohvaćanje preporuka za korisnika ID: {UserId}", userId);

var hasOrders = await context.Orders.AnyAsync(o => o.UserId == userId);

if (!hasOrders)

{

\_logger.LogInformation("Korisnik ID: {UserId} nema narudžbe, vraćamo featured proizvode", userId);

return await GetFeaturedProductsAsync(maxResults);

}

var lastOrders = await context.Orders

.Where(o => o.UserId == userId)

.OrderByDescending(o => o.OrderDate)

.Include(o => o.OrderDetails)

.Take(3)

.ToListAsync();

var recommendedIds = new HashSet<int>();

foreach (var order in lastOrders)

{

foreach (var item in order.OrderDetails.Where(od => od.ProductId.HasValue))

{

var recommendations = await GetRecommendedProductsAsync(item.ProductId.Value, 3);

foreach (var product in recommendations)

recommendedIds.Add(product.Id);

if (!recommendedIds.Contains(item.ProductId.Value))

recommendedIds.Add(item.ProductId.Value);

}

}

var recommendedProducts = await context.Products

.Where(p => recommendedIds.Contains(p.Id) && p.IsAvailable && p.Active)

.Include(p => p.Images)

.Include(p => p.Category)

.Include(p => p.Occasion)

.Take(maxResults)

.ToListAsync();

\_logger.LogInformation("Korisnik ID: {UserId} ima {OrderCount} narudžbi, preporučuje se {ProductCount} proizvoda", userId, lastOrders.Count, recommendedProducts.Count);

var result = recommendedProducts.Select(p => new ProductResponse

{

Id = p.Id,

Name = p.Name,

Description = p.Description,

Price = p.Price,

IsNew = p.IsNew,

IsFeatured = p.IsFeatured,

CategoryId = p.CategoryId,

CategoryName = p.Category?.Name,

OccasionId = p.OccasionId,

OccasionName = p.Occasion?.Name,

Active = p.Active,

IsAvailable = p.IsAvailable,

ImageUrls = p.Images.Select(i => i.ImageUrl).ToList()

}).ToList();

return result;

}

catch (Exception ex)

{

\_logger.LogError(ex, "Greška prilikom dohvaćanja preporučenih proizvoda za korisnika ID: {UserId}", userId);

return new List<ProductResponse>();

}

}

private async Task<List<ProductResponse>> GetFeaturedProductsAsync(int count)

{

using var context = \_contextFactory.CreateDbContext();

var featuredProducts = await context.Products

.Where(p => p.IsAvailable && p.Active && (p.IsFeatured || p.IsNew))

.Include(p => p.Images)

.Include(p => p.Category)

.Include(p => p.Occasion)

.Take(count)

.ToListAsync();

if (featuredProducts.Count == 0)

{

featuredProducts = await context.Products

.Where(p => p.IsAvailable && p.Active)

.Include(p => p.Images)

.Include(p => p.Category)

.Include(p => p.Occasion)

.OrderBy(p => p.Id)

.Take(count)

.ToListAsync();

}

return featuredProducts.Select(p => new ProductResponse

{

Id = p.Id,

Name = p.Name,

Description = p.Description,

Price = p.Price,

IsNew = p.IsNew,

IsFeatured = p.IsFeatured,

CategoryId = p.CategoryId,

CategoryName = p.Category?.Name,

OccasionId = p.OccasionId,

OccasionName = p.Occasion?.Name,

Active = p.Active,

IsAvailable = p.IsAvailable,

ImageUrls = p.Images.Select(i => i.ImageUrl).ToList()

}).ToList();

}

}

## Putanja i kod glavne logike kontrolera sistema preporuka

**Putanja:** FloraApp\_RS2\Flora backend\FloraAPI\Controllers\RecommendationsController

[HttpGet("products/{productId}")]

public async Task<ActionResult<List<ProductResponse>>> GetProductRecommendations(int productId, [FromQuery] int topN = 5)

{

try

{

var recommendations = await \_recommendationService.GetRecommendedProductsAsync(productId, topN);

\_logger.LogInformation("Dohvaćeno {Count} preporuka za proizvod ID: {ProductId}", recommendations.Count, productId);

return Ok(recommendations);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Greška prilikom dohvaćanja preporuka za proizvod ID: {ProductId}", productId);

return StatusCode(StatusCodes.Status500InternalServerError, "Došlo je do greške prilikom dohvaćanja preporuka");

}

}

[HttpPost("recalculate")]

public async Task<IActionResult> RecalculateRecommendations()

{

try

{

await \_recommendationService.RecalculateSimilarityMapAsync();

return Ok("Izračun sličnosti proizvoda je uspješno pokrenut");

}

catch (Exception ex)

{

\_logger.LogError(ex, "Greška prilikom izračuna sličnosti proizvoda");

return StatusCode(StatusCodes.Status500InternalServerError, "Došlo je do greške prilikom izračuna sličnosti proizvoda");

}

}

[HttpGet("co-purchases")]

public async Task<ActionResult<List<ProductCoPurchase>>> GetCoPurchaseMap()

{

try

{

var coPurchaseMap = await \_recommendationService.GetCoPurchaseMapAsync();

return Ok(coPurchaseMap);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Greška prilikom dohvaćanja mape ko-kupovina");

return StatusCode(StatusCodes.Status500InternalServerError, "Došlo je do greške prilikom dohvaćanja mape ko-kupovina");

}

}

[HttpGet("user/{userId}")]

public async Task<ActionResult<List<ProductResponse>>> GetRecommendationsForUser(int userId, [FromQuery] int maxResults = 10)

{

try

{

var recommendations = await \_recommendationService.GetRecommendedForUserAsync(userId, maxResults);

\_logger.LogInformation("Dohvaćeno {Count} preporuka za korisnika ID: {UserId}", recommendations.Count, userId);

return Ok(recommendations);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Greška prilikom dohvaćanja preporuka za korisnika ID: {UserId}", userId);

return StatusCode(StatusCodes.Status500InternalServerError, "Došlo je do greške prilikom dohvaćanja preporuka za korisnika");

}

}

## Putanja i printscreen iz pokrenute aplikacije gdje se prikazuju preporučeni proizvodi

U mobile aplikaciji (username-mobile, password-test), na početnoj stranici u četvrtoj sekciji se nalaze preporučeni proizvodi

A screenshot of a phone

Description generated with very high confidence