1. Opis implementacije recommender sistema

U aplikaciji je korišten **content-based filtering**. Cilj je korisniku preporučiti proizvode slične onima koje je već kupio. Ova metoda se zasniva na analiziranju karakteristika svakog proizvoda i formiranju vektorske reprezentacije kojom se mjeri sličnost između proizvoda.

Podaci se sastoje iz dvije glavne grupe:

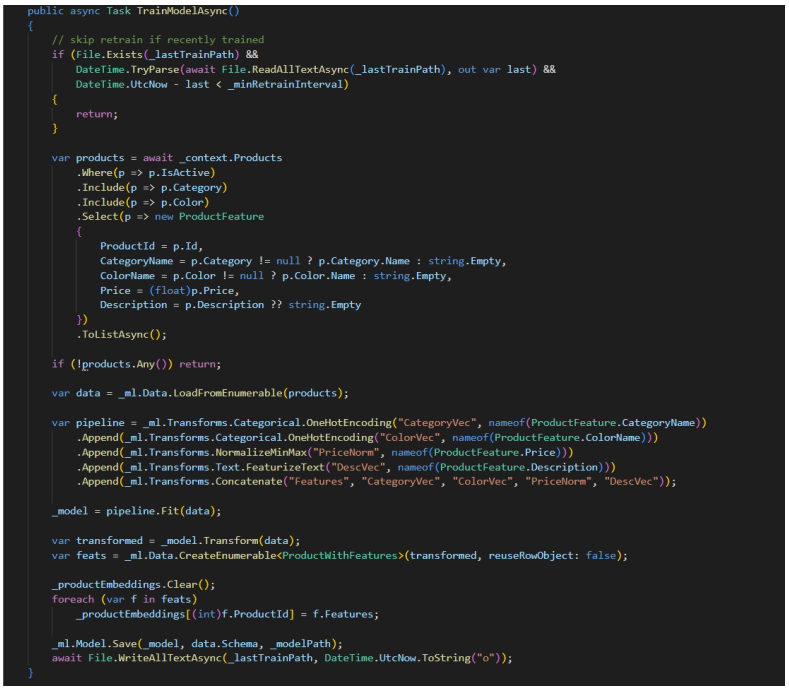
- **Korisnička historija** – proizvodi koje je korisnik kupio putem narudžbi
- **Karakteristike proizvoda** – atributi kao što su kategorija, boja, cijena i opis

ML.NET se koristi za ekstrakciju karakteristika: kategorija i boja kodiraju se OneHot tehnikom, cijena se normalizuje, a opis proizvoda prolazi kroz featurizaciju teksta. Svi atributi se spajaju u jedan **Features vektor** koji predstavlja proizvod u višedimenzionalnom prostoru.

Za formiranje korisničkog profila uzima se aritmetička sredina vektora proizvoda koje je korisnik već kupio. Svaki proizvod iz baze koji korisnik još nije kupio poredi se sa ovim profilom pomoću **euklidske udaljenosti**. Manja udaljenost znači veću sličnost, a sistem vraća listu top-N najbližih proizvoda.

2. Putanja i printscreen source code-a glavne logike recommender Sistema

Putanja: "MyClub\MyClub.Services\Services\RecommendationService.cs"



```
public async Task TrainModelAsync()
{
    // skip retrain if recently trained
    if (File.Exists(_lastTrainPath) &&
        DateTime.TryParse(await File.ReadAllTextAsync(_lastTrainPath), out var last) &&
        DateTime.UtcNow - last < _minRetrainInterval)
    {
        return;
    }

    var products = await _context.Products
        .Where(p => p.IsActive)
        .Include(p => p.Category)
        .Include(p => p.Color)
        .Select(p => new ProductFeature
        {
            ProductId = p.Id,
            CategoryName = p.Category != null ? p.Category.Name : string.Empty,
            ColorName = p.Color != null ? p.Color.Name : string.Empty,
            Price = (float)p.Price,
            Description = p.Description ?? string.Empty
        })
        .ToListAsync();

    if (!products.Any()) return;

    var data = _ml.Data.LoadFromEnumerable(products);

    var pipeline = _ml.Transforms.Categorical.OneHotEncoding("CategoryVec", nameof(ProductFeature.CategoryName))
        .Append(_ml.Transforms.Categorical.OneHotEncoding("ColorVec", nameof(ProductFeature.ColorName)))
        .Append(_ml.Transforms.NormalizeMinMax("PriceNorm", nameof(ProductFeature.Price)))
        .Append(_ml.Transforms.Text.FeaturizeText("DescVec", nameof(ProductFeature.Description)))
        .Append(_ml.Transforms.Concatenate("Features", "CategoryVec", "ColorVec", "PriceNorm", "DescVec"));

    _model = pipeline.Fit(data);

    var transformed = _model.Transform(data);
    var feats = _ml.Data.CreateEnumerable<ProductWithFeatures>(transformed, reuseRowObject: false);

    _productEmbeddings.Clear();
    foreach (var f in feats)
        _productEmbeddings[(int)f.ProductId] = f.Features;

    _ml.Model.Save(_model, data.Schema, _modelPath);
    await File.WriteAllTextAsync(_lastTrainPath, DateTime.UtcNow.ToString("o"));
}
```

```
public async Task<List<ProductResponse>> GetRecommendationsAsync(int userId, int count = 10)
{
    if (_model == null || !_productEmbeddings.Any())
        throw new InvalidOperationException("Model not trained");

    var purchasedIds = await _context.Orders
        .Where(o => o.UserId == userId)
        .SelectMany(o => o.OrderItems)
        .Select(oi => oi.ProductSize.ProductId)
        .Distinct()
        .ToListAsync();

    if (!purchasedIds.Any())
        return new List<ProductResponse>(); // caller should fallback to newest

    var purchasedVectors = purchasedIds.Where(id => _productEmbeddings.ContainsKey(id))
                                       .Select(id => _productEmbeddings[id])
                                       .ToList();

    if (!purchasedVectors.Any())
        return new List<ProductResponse>();

    int dim = purchasedVectors.First().Length;
    var userProfile = new float[dim];
    foreach (var v in purchasedVectors)
        for (int i = 0; i < dim; i++)
            userProfile[i] += v[i];
    for (int i = 0; i < dim; i++)
        userProfile[i] /= purchasedVectors.Count;

    var distances = new List<(int ProductId, double Distance)>();
    foreach (var kv in _productEmbeddings)
    {
        int pid = kv.Key;
        if (purchasedIds.Contains(pid)) continue;

        double dist = EuclideanDistance(userProfile, kv.Value);
        distances.Add((pid, dist));
    }

    var topIds = distances.OrderBy(d => d.Distance) // ascending = closest
                          .Take(count)
                          .Select(d => d.ProductId)
                          .ToList();

    if (!topIds.Any()) return new List<ProductResponse>();

    var products = await _context.Products
        .Where(p => topIds.Contains(p.Id) && p.IsActive)
        .Include(p => p.Category)
        .Include(p => p.Color)
        .Include(p => p.ProductAssets)
        .ThenInclude(pa => pa.Asset)
        .Include(p => p.ProductSizes)
        .ThenInclude(ps => ps.Size)
        .ToListAsync();

    // preserve ordering by topIds
    var ordered = topIds.Select(id => products.First(p => p.Id == id)).ToList();
    return ordered.Select(p =>  mapper.Map<ProductResponse>(p)).ToList();
```

3. Putanja i printscreen iz pokrenute aplikacije gdje se prikazuju preporuke

Putanja: "MyClub\UI\myclub_mobile\lib\screens\shop_screen.dart"

Printscreen: