Sistem preporuka 3D modela

1. Opis implementacije recommender sistema

JoyModels platforma implementira personalizirani sistem preporuka baziran na mašinskom učenju, čija je svrha predviđanje koje 3D modele bi određeni korisnik mogao poželjeti kupiti. Sistem koristi kolaborativno filtriranje putem algoritma matrične faktorizacije iz Microsoft ML.NET biblioteke, tehniku koja se koristi u modernim sistemima preporuka poput Netflixa i Amazona.

Prikupljanje podataka

Sistem prikuplja korisničke interakcije sa različitim težinskim faktorima koji odražavaju snagu korisničkog interesa:

Tip interakcije	Score	Obrazloženje
Kupovine	1.0	Najjača indikacija interesa, korisnik je izdvojio novac za model
Lajkovi	0.7	Pozitivan stav bez finansijske obaveze
Stavke u košarici	0.4	Namjera kupovine koja nije završena
Negativni uzorci	0.1	Nasumični modeli bez interakcije, omogućava razlikovanje preferiranih od nepoželjnih

Algoritam matrične faktorizacije

Algoritam radi tako što analizira ponašanje svih korisnika i na osnovu toga pronalazi skrivene obrasce, npr. koji tipovi korisnika preferiraju koje tipove modela. Kada sistem treba preporučiti modele određenom korisniku, on predvidi koliko bi se tom korisniku svidio svaki model i ponudi one sa najvišim ocjenama.

Model se trenira pri pokretanju aplikacije i automatski se ponovno trenira svakih 6 sati kako bi uključio nove podatke o ponašanju korisnika. Istrenirani

model se sprema na disk kako bi se pri ponovnom pokretanju aplikacije mogao brzo učitati bez potrebe za ponovnim treniranjem.

Generisanje preporuka
Kada korisnik zatraži preporuke, sistem prolazi kroz sljedeće korake:

1. Provjerava da li je model istreniran i da li korisnik postoji u training setu
2. Dohvata sve javne 3D modele i eliminiše one sa kojima je korisnik već imao interakciju (kupovine, lajkovi, košarica)
3. Za svakog kandidata računa predviđeni score koristeći prediction engine
4. Sortira modele po score-u i vraća traženi broj najrelevantnijih rezultata
5. Za nove korisnike koji nisu u training setu koristi fallback strategiju tj. vraća najprodavanije modele

Health Check integracija
Sistem je integrisan sa ASP.NET Core Health Checks mehanizmom. Endpoint /health vraća status "Healthy" samo kada se migracije uspješno završe, baza podataka napuni dummy podacima i model uspješno istrenira ili učita sa diska. Ovo omogućava Docker kontejner orkestraciji da zna kada je aplikacija potpuno spremna za primanje korisničkog traffica.

2. Putanje i printscreen source code-a glavne logike

Glavna logika recommender sistema nalazi se u sljedećim datotekama:

Treniranje, predikcije i persistencija modela:
JoyModels.Services/src/Services/Recommender/RecommenderService.cs

Interface servisa:
JoyModels.Services/src/Services/Recommender/IRecommenderService.cs

Pozadinski servis za životni ciklus modela:
JoyModels.API/src/Services/RecommenderTrainingService.cs

Health Check:

JoyModels.API/src/Handlers/RecommenderHealthCheck.cs

```
0+1 usages   Faris
public async Task TrainModel()
{
    await using var context = await contextFactory.CreateDbContextAsync();

    var newUserIdMap = new Dictionary<Guid, uint>();
    var newModelIdMap = new Dictionary<Guid, uint>();

    var trainingData :List<ModelInteractionRequest> = await GetTrainingDataAsync(context, newUserIdMap, newModelIdMap);

    if (trainingData.Count < 10)
    {
        return;
    }

    var dataView = _mlContext.Data.LoadFromEnumerable(trainingData);

    var options = new MatrixFactorizationTrainer.Options
    {
        MatrixColumnIndexColumnName = nameof(ModelInteractionRequest.UserId),
        MatrixRowIndexColumnName = nameof(ModelInteractionRequest.ModelId),
        LabelColumnName = nameof(ModelInteractionRequest.Score),
        NumberOfIterations = 20,
        ApproximationRank = 32,
        LearningRate = 0.1
    };

    var trainer = _mlContext.Recommendation().Trainers.MatrixFactorization(options);
    var trainedModel :MatrixFactorizationPredictionTransformer? = trainer.Fit(dataView);
    var predictionEngine =
        _mlContext.Model.CreatePredictionEngine<ModelInteractionRequest, ModelInteractionResponse>(trainedModel);

    lock (_lock)
    {
        _trainedModel = trainedModel;
        _predictionEngine = predictionEngine;
        _userIdMap = newUserIdMap;
        _modelIdMap = newModelIdMap;
    }

    SaveModel(dataView.Schema, newUserIdMap, newModelIdMap);
}
```

```
0+1 usages  Faris
public async Task<List<Guid>> GetRecommendations(Guid userUuid, int count)
{
    await using var context = await contextFactory.CreateDbContextAsync();

    uint userId = 0;
    Dictionary<Guid, uint>? modelIdMapSnapshot = null;
    bool useFallback;

    lock (_lock)
    {
        if (_trainedModel == null || _predictionEngine == null)
        {
            return [];
        }

        if (!_userIdMap.TryGetValue(userUuid, out userId))
        {
            useFallback = true;
        }
        else
        {
            useFallback = false;
            modelIdMapSnapshot = new Dictionary<Guid, uint>(_modelIdMap);
        }
    }

    if (useFallback)
    {
        return await GetFallbackRecommendationsAsync(context, userUuid, count);
    }

    var userInteractedModelUuids :HashSet<Guid> = await GetUserInteractedModelsAsync(context, userUuid);

    var publicModelUuids :List<Guid> = await context.Models // DbSet<Model>
        .AsNoTracking()
        .Where(m :Model => m.ModelAvailabilityUu.AvailabilityName == nameof(ModelAvailabilityEnum.Public)) // IQueryable<Model>
        .Select(m :Model => m.Uuid) // IQueryable<Guid>
        .ToListAsync(); // Task<List<...>>

    var candidateModels :List<Guid> = publicModelUuids // List<Guid>
        .Except(userInteractedModelUuids) // IEnumerable<Guid>
        .ToList();

    var predictions = new List<(Guid ModelUuid, float Score)>();
```

```
    var predictions = new List<(Guid ModelUuid, float Score)>();

    lock (_lock)
    {
        if (_predictionEngine == null)
        {
            return new List<Guid>();
        }

        foreach (var modelUuid :Guid in candidateModels)
        {
            if (!modelIdMapSnapshot!.TryGetValue(modelUuid, out var modelId :uint))
                continue;

            var prediction :ModelInteractionResponse? = _predictionEngine.Predict(new ModelInteractionRequest
            {
                UserId = userId,
                ModelId = modelId
            });

            if (!float.IsNaN(prediction.Score))
            {
                predictions.Add((modelUuid, prediction.Score));
            }
        }
    }

    var result :List<Guid> = predictions // List<(ModelUuid,Score)>
        .OrderByDescending(p :(ModelUuid,Score) => p.Score) // IOrderedEnumerable<(ModelUuid,Score)>
        .Take(count) // IEnumerable<(ModelUuid,Score)>
        .Select(p :(ModelUuid,Score) => p.ModelUuid) // IEnumerable<Guid>
        .ToList();

    if (result.Count == 0)
    {
        return await GetFallbackRecommendationsAsync(context, userUuid, count);
    }

    return result;
}
```

```
2 usages  Faris
public class RecommenderTrainingService(
    IRecommenderService recommenderService,
    ILogger<RecommenderTrainingService> logger)
    : BackgroundService
{
    private readonly TimeSpan _trainingInterval = TimeSpan.FromHours(6);

    Faris
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        if (!TryLoadModel())
        {
            await TrainModelAsync();
        }

        using var timer = new PeriodicTimer(_trainingInterval);

        while (!stoppingToken.IsCancellationRequested && await timer.WaitForNextTickAsync(stoppingToken))
        {
            await TrainModelAsync();
        }
    }

    1 usage  Faris
    private bool TryLoadModel()
    {
        logger.LogInformation("Attempting to load recommender model from disk...");

        var loaded :bool = recommenderService.LoadModel();

        if (loaded)
        {
            logger.LogInformation("Recommender model loaded successfully from disk.");
        }
        else
        {
            logger.LogInformation("No saved model found. Will train a new model.");
        }

        return loaded;
    }

    2 usages  Faris
    private async Task TrainModelAsync()
    {
        try
        {
            logger.LogInformation("Starting recommender model training...");
            await recommenderService.TrainModel();

            if (recommenderService.IsModelTrained)
            {
                logger.LogInformation("Recommender model training completed successfully.");
            }
            else
            {
                logger.LogWarning(
                    "Recommender model training skipped - insufficient data (need at least 10 interactions).");
            }
        }
        catch (Exception ex)
        {
            logger.LogError(ex, message: "Error during recommender model training.");
        }
    }
}
```

3. Putanje i printscreen iz pokrenute aplikacije

Preporuke se prikazuju korisniku na početnoj stranici aplikacije nakon prijave.