Bioskopina - Recommender Sistem

Inspiracija i primjer – Netflix

Glavna inspiracija za razvoj sistema bila je platforma **Netflix**, koja koristi slične algoritme kako bi preporučila korisnicima nove filmove i serije na temelju njihovih prethodnih aktivnosti. Netflix je poznat po sofisticiranom preporučivačkom mehanizmu koji koristi podatke o gledanju i ocjenama korisnika, te odnose između sadržaja koje korisnici konzumiraju. Na sličan način, ovaj sistem koristi podatke iz korisničkih listi filmova unutar aplikacije da bi pronašao povezanosti između filmova koji se često pojavljuju zajedno.

Opis implementacije recommender sistema

U okviru ovog projekta implementiran je **preporučivački sistem** temeljen na metodi **faktorizacije matrica (Matrix Factorization)** koristeći **ML.NET** biblioteke za mašinsko učenje. Ova metoda spada u grupu **sistema za preporuku baziranih na suradnji (collaborative filtering)** i koristi prethodno ponašanje korisnika kao osnovu za predviđanje sličnih sadržaja. Sistemu se pristupa putem baze podataka u kojoj se nalaze korisničke liste filmova. Ako se dva filma pojavljuju zajedno na više različitih lista, to se tretira kao znak povezanosti među njima.

Print screen:

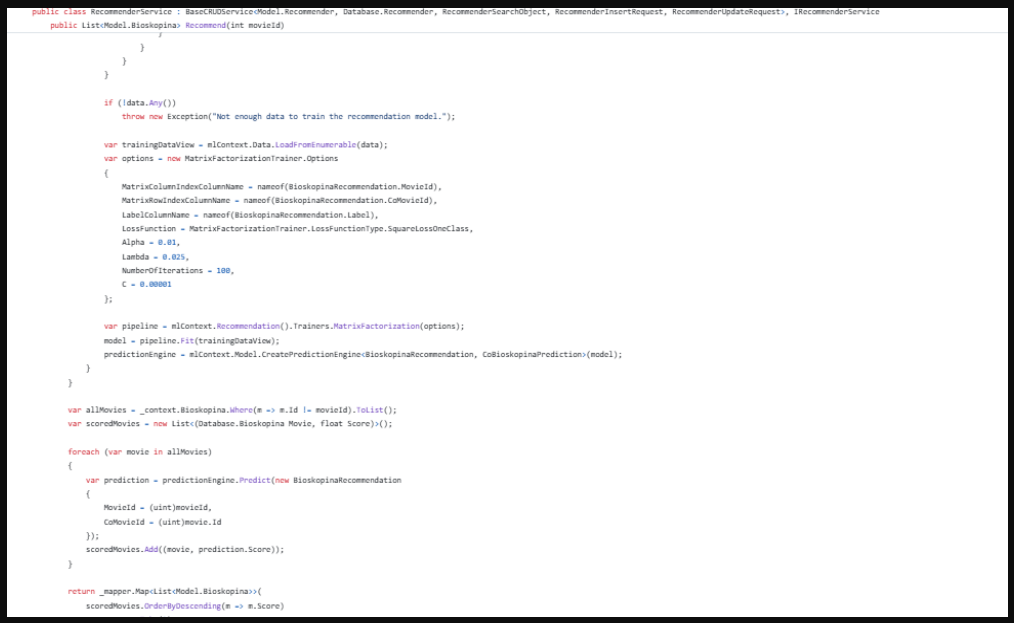
```
public async Task<Model.Recommender?> GetById(int movieId, CancellationToken cancellationToken = default)
{
    var entity = await _context.Recommenders.FirstOrDefaultAsync(r => r.MovieId == movieId, cancellationToken);
    return entity == null ? null : _mapper.Map<Model.Recommender>(entity);
}

public List<Model.Bioskopina> Recommend(int movieId)
{
    lock (isLocked)
    {
        if (mlContext == null || model == null)
        {
            mlContext = new MLContext();
            var lists = _context.Lists
                .Include(l => l.BioskopinaLists)
                .Where(l => l.BioskopinaLists.Count > 1)
                .ToList();

            var data = new List<BioskopinaRecommendation>();

            foreach (var list in lists)
            {
                var movieIds = list.BioskopinaLists.Select(bl => bl.MovieId).Distinct().ToList();
                foreach (var movieA in movieIds)
                {
                    foreach (var movieB in movieIds)
                    {
                        if (movieA != movieB)
                        {
                            data.Add(new BioskopinaRecommendation
                            {
                                MovieId = (uint)movieA,
                                CoMovieId = (uint)movieB,
                                Label = 1f
                            });
                        }
                    }
                }
            }

            if (!data.Any())
```



```
public class RecommenderService : BaseCRUDService<Model.Recommender, Database.Recommender, RecommenderSearchObject, RecommenderInsertRequest, RecommenderUpdateRequest>, IRecommenderService
    public List<Model.Bioskopina> Recommend(int movieId)
                    }
                }
            }

            if (!data.Any())
                throw new Exception("Not enough data to train the recommendation model.");

            var trainingDataView = mlContext.Data.LoadFromEnumerable(data);
            var options = new MatrixFactorizationTrainer.Options
            {
                MatrixColumnIndexColumnName = nameof(BioskopinaRecommendation.MovieId),
                MatrixRowIndexColumnName = nameof(BioskopinaRecommendation.CoMovieId),
                LabelColumnName = nameof(BioskopinaRecommendation.Label),
                LossFunction = MatrixFactorizationTrainer.LossFunctionType.SquareLossOneClass,
                Alpha = 0.01,
                Lambda = 0.025,
                NumberOfIterations = 100,
                C = 0.00001
            };

            var pipeline = mlContext.Recommendation().Trainers.MatrixFactorization(options);
            model = pipeline.Fit(trainingDataView);
            predictionEngine = mlContext.Model.CreatePredictionEngine<BioskopinaRecommendation, CoBioskopinaPrediction>(model);
        }
    }

    var allMovies = _context.Bioskopina.Where(m => m.Id != movieId).ToList();
    var scoredMovies = new List<(Database.Bioskopina Movie, float Score)>();

    foreach (var movie in allMovies)
    {
        var prediction = predictionEngine.Predict(new BioskopinaRecommendation
        {
            MovieId = (uint)movieId,
            CoMovieId = (uint)movie.Id
        });
        scoredMovies.Add((movie, prediction.Score));
    }

    return _mapper.Map<List<Model.Bioskopina>>(
        scoredMovies.OrderByDescending(m => m.Score)
```

Putanja: Bioskopina/ backend/Bioskopina.Services/RecommenderService.cs

UI aplikacije:

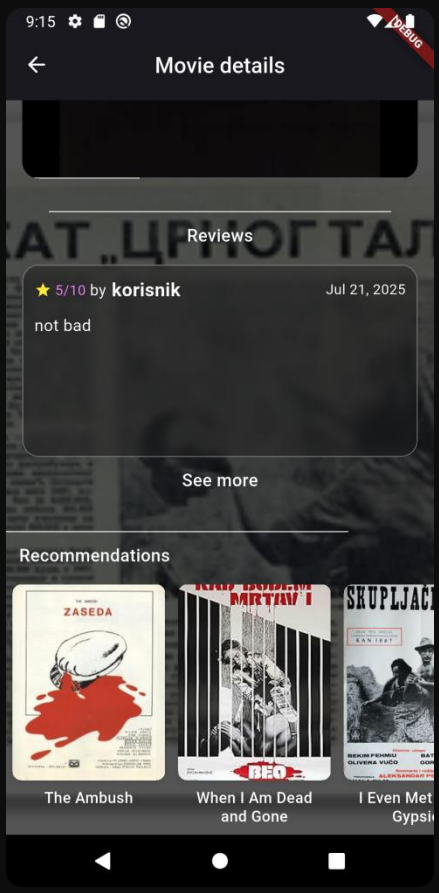
Recommender sistem se skrola horizonalno, glavna referenca za ovaj dizajn jeste:

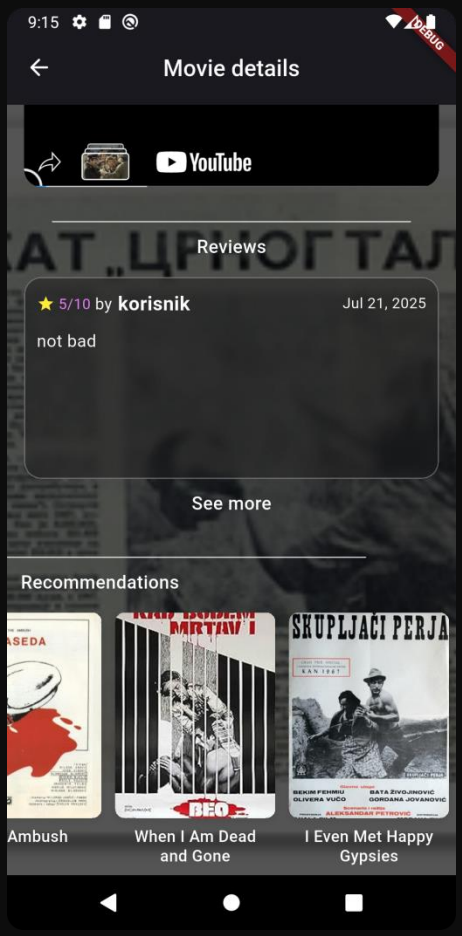
Horizontal Scrolling Lists in Mobile - Best Practices

Inspiracija:

screen696x696.jpeg (392×696)

amazon_prime_video_android.png (360×640)





Putanja:

Bioskopina/ bioskopina_mobile/lib/screens/bioskopina_detail_screen.dart