Nejla Čajdin

Razvoj softvera 2

21.8.2025.

Implementacija recommender sistema

BookWorm je društvena aplikacija za knjige koja korisnicima omogućava da vode svoje liste pročitanih i željenih knjiga, učestvuju u čitalačkim klubovima i povezuju se sa drugim ljubiteljima knjiga. Kako aplikacija raste i broj knjiga i korisnika postaje sve veći, korisnicima može biti teško pronaći nove knjige koje bi im se svidjele ili otkriti osobe sa sličnim interesima. Zato je **recommender sistem ključna funkcionalnost** – on personalizira iskustvo svakog korisnika tako što predlaže knjige koje bi vjerovatno želio čitati i potencijalne prijatelje sa kojima dijeli slične čitalačke navike.

Unutar ove aplikacije implementirana su dva sistema preporuke: Preporuka knjiga i preporuka prijatelja (korisnika).

Preporuka knjiga

Da bi korisnik dobio personalizovane preporuke, prvo mora da označi knjige koje je pročitao (dodavanjem u listu **Read**). Sistem zatim uzima sve knjige iz te liste i poredi ih sa ostalim knjigama u bazi. To jeste gleda koje se knjige cesto pojavljuju sa korisnikovim knjigama u listi **Read** (koje knjige se često čitaju zajedno). Trenirani model koristi matrix factorization da izračuna koliko je neka druga knjiga „slična“ onoj koju je korisnik pročitao. Za svaku kandidat knjigu računa se score (ocjena sličnosti). Ako se knjiga pojavi više puta kroz poređenje, score se sabira.Na kraju se sve knjige sortiraju po ukupnom score-u i korisnik dobija listu top preporuka (npr. 15 najboljih).Ako korisnik još nije pročitao nijednu knjigu, sistem mu jednostavno ponudi najpopularnije knjige (one koje su najviše čitane od drugih). Kada se model istrenira, on se sprema u fajl **model.zip**. Kod svakog novog pokretanja aplikacije, sistem prvo provjerava da li taj fajl postoji. Ako postoji, model se samo učita iz fajla i koristi odmah. Ako ne postoji, pokreće se treniranje od nule i model se ponovo snimi.

Putanja do source code-a:

BookWorm\BookWorm\BookWorm.Services\BookService.cs

Printscreenovi book recommender logike

```
2 references | nejlac, 1 day ago | 1 author, 3 changes
public async Task<List<BookResponse>> GetRecommendedBooksAsync(int userId)
{
    if (_model == null || _predictionEngine == null)
        await LoadOrTrainModelAsync();

    var readBooks = await _context.ReadingListBooks
        .Where(rlb => rlb.ReadingList.UserId == userId && rlb.ReadingList.Name == "Read")
        .Select(rlb => rlb.BookId)
        .Distinct()
        .ToListAsync();

    if (!readBooks.Any())
    {
        //if user doesnt have any books in "Read" list, he gets the most popular ones suggested
        return await GetMostReadBooks(15).ContinueWith(t => t.Result.Select(x => new BookResponse
        {
            Id = x.BookId,
            Title = x.Title,
            AuthorName = x.AuthorName,
            CoverImagePath = x.CoverImageUrl
        }).ToList());
    }

    var allBooks = await _context.Books.ToListAsync();
    var scores = new Dictionary<int, float>();

    foreach (var readBookId in readBooks)
    {
        foreach (var book in allBooks)
        {
            // Skip if this is the same book or if user has already read it
            if (readBookId == book.Id || readBooks.Contains(book.Id))
                continue;

            var prediction = _predictionEngine!.Predict(new BookEntry
            {
                BookId = (uint)readBookId,
                CoReadBookId = (uint)book.Id
            });

            if (scores.ContainsKey(book.Id))
                scores[book.Id] += prediction.Score;
            else
                scores[book.Id] = prediction.Score;
        }
    }
```

```
    var recommendedBookIds = scores.OrderByDescending(x => x.Value).Take(15).Select(x => x.Key).ToList();

    var recommendedBooks = await _context.Books
        .Where(b => recommendedBookIds.Contains(b.Id))
        .Include(b => b.Author)
        .ToListAsync();

    return recommendedBooks.Select(b => MapToResponse(b)).ToList();
}

1 reference | nejlac, 18 days ago | 1 author, 1 change
private async Task LoadOrTrainModelAsync()
{
    if (File.Exists(_modelFilePath))
    {
        using var stream = new FileStream(_modelFilePath, FileMode.Open, FileAccess.Read, FileShare.Read);
        _model = _mlContext.Model.Load(stream, out var schema);
        _predictionEngine = _mlContext.Model.CreatePredictionEngine<BookEntry, BookPrediction>(_model);
    }
    else
    {
        await TrainAndSaveModelAsync();
    }
}
```

```
private async Task TrainAndSaveModelAsync()
{
    var readLists = await _context.ReadingLists
        .Where(rl => rl.Name == "Read")
        .Include(rl => rl.ReadingListBooks)
        .ToListAsync();

    var data = new List<BookEntry>();

    foreach (var rl in readLists)
    {
        var bookIds = rl.ReadingListBooks.Select(x => x.BookId).Distinct().ToList();

        foreach (var b1 in bookIds)
        {
            foreach (var b2 in bookIds)
            {
                if (b1 != b2)
                {
                    data.Add(new BookEntry
                    {
                        BookId = (uint)b1,
                        CoReadBookId = (uint)b2,
                        Label = 1
                    });
                }
            }
        }
    }

    if (!data.Any())
        return;

    var traindata = _mlContext.Data.LoadFromEnumerable(data);

    var options = new MatrixFactorizationTrainer.Options
    {
        MatrixColumnIndexColumnName = nameof(BookEntry.BookId),
        MatrixRowIndexColumnName = nameof(BookEntry.CoReadBookId),
        LabelColumnName = nameof(BookEntry.Label),
        NumberOfIterations = 100,
        ApproximationRank = 32,
        Alpha = 0.01,
        Lambda = 0.025,
        LossFunction = MatrixFactorizationTrainer.LossFunctionType.SquareLossOneClass,
        C = 0.00001
    };

    var estimator = _mlContext.Recommendation().Trainers.MatrixFactorization(options);
    _model = estimator.Fit(traindata);

    using var fs = new FileStream(_modelFilePath, FileMode.Create, FileAccess.Write, FileShare.Write);
    _mlContext.Model.Save(_model, traindata.Schema, fs);

    _predictionEngine = _mlContext.Model.CreatePredictionEngine<BookEntry, BookPrediction>(_model);
}
```

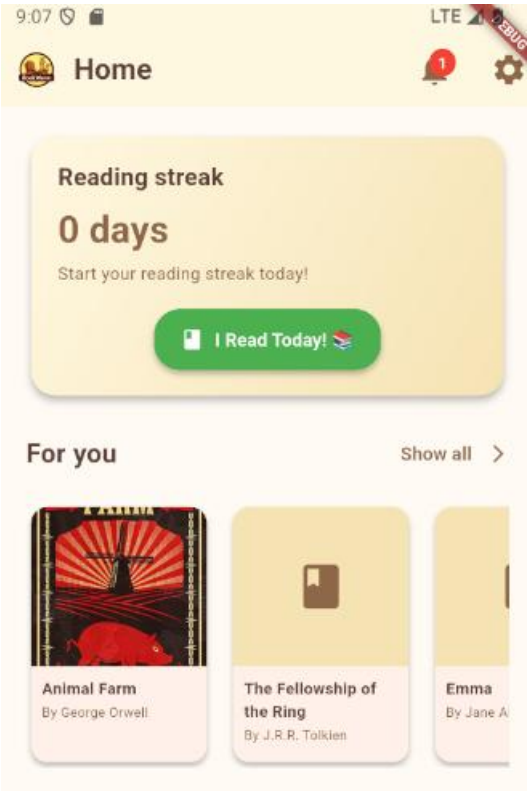
Putanja do code-a u aplikaciji gdje se poziva recommender sistem:

BookWorm\BookWorm\UI\bookworm_mobile\lib\screens\homepage.dart

U aplikaciji BookWorm preporuke se prikazuju na početnoj stranici nakon što se korisnik uloguje. Tu se prikazuju predložene knjige. Ako je model već istreniran, preporuke se odmah učitavaju iz fajla (model.zip), a u suprotnom se model trenira i zatim koristi za prikaz personalizovanih rezultata. Ukoliko korisnik nema pročitanih knjiga (prazna lista **Read**), učitat će mu se najčitanije knjige.

Printscreen iz pokrenute aplikacije

For you sekcija se odnosi na preporučene knjige



Preporuka prijatelja

Da bi preporuke imale smisla, korisnik opet mora imati označene knjige koje je pročitao. Sistem trenira model da uoči koje osobe imaju slične čitalačke navike (imaju najviše zajedničkih knjiga pročitanih). Kada se traže prijatelji, za svakog drugog korisnika računa se **score sličnosti** (na osnovu zajednički pročitanih knjiga). Rezultat je lista korisnika sa najvećim score-om. Ako trenutni korisnik nema pročitanih knjiga, onda se kao fallback nude **najaktivniji korisnici** (oni sa najviše pročitanih knjiga).Već postojeći prijatelji se automatski isključuju iz liste. Ovdje se također treniran model sprema u fajl **ml_friend_model.zip**, te se pri učitavanju iz njega čitaju podaci (ukoliko postoji). Razlikuje se od prethodnog spašavanja po tome što se ovo treniranje modela dešava svaki dan.

Putanja do source code-a:

BookWorm\BookWorm\BookWorm.Services\UserService.cs

Printscreenovi recommender logike

```
2 references | nejlac, 16 days ago | 1 author, 2 changes
public async Task<List<UserResponse>> RecommendFriends(int userId)
{
    var mlContext = new MLContext();

    var userReadMap = await _context.ReadingLists
        .Where(rl => rl.Name == "Read")
        .Include(rl => rl.ReadingListBooks)
        .ToListAsync();

    var data = new List<UserBookEntry>();

    foreach (var list in userReadMap)
    {
        var distinctBookIds = list.ReadingListBooks.Select(r => r.BookId).Distinct().ToList();

        foreach (var bookId in distinctBookIds)
        {
            foreach (var otherBookId in distinctBookIds.Where(id => id != bookId))
            {
                data.Add(new UserBookEntry
                {
                    UserId = (uint)list.UserId,
                    BookId = (uint)bookId,
                    Label = 1f
                });
            }
        }
    }

    if (data.Count == 0)
    {
        // if there are no books read by current user, fallback to most active users
        return await GetMostActiveUsersFallback(userId);
    }

    ITransformer? model = null ;
    var modelPath = "ml_friend_model.zip";
    bool retrainModel = true;

    if (File.Exists(modelPath))
    {
        var lastWriteTime = File.GetLastWriteTime(modelPath);
        if ((DateTime.Now - lastWriteTime).TotalDays < 1)
        {
            using var fileStream = new FileStream(modelPath, FileMode.Open, FileAccess.Read, FileShare.Read);
            model = mlContext.Model.Load(fileStream, out var _);
            retrainModel = false;
        }
```

```
if (retrainModel)
{
var trainData = mlContext.Data.LoadFromEnumerable(data);

var options = new MatrixFactorizationTrainer.Options
{
    MatrixColumnIndexColumnName = nameof(UserBookEntry.BookId),
    MatrixRowIndexColumnName = nameof(UserBookEntry.UserId),
    LabelColumnName = nameof(UserBookEntry.Label),
    LossFunction = MatrixFactorizationTrainer.LossFunctionType.SquareLossOneClass,
    Alpha = 0.01,
    Lambda = 0.1,
    NumberOfIterations = 40,
    C = 0.0001
};

        var estimator = mlContext.Recommendation().Trainers.MatrixFactorization(options);
model = estimator.Fit(trainData);

using var fileStream = new FileStream(modelPath, FileMode.Create, FileAccess.Write, FileShare.Write);
mlContext.Model.Save(model, trainData.Schema, fileStream);
}


// Get users who are already friends with the current user (either as sender or receiver)
var existingFriendships = await _context.UserFriends
    .Where(uf => (uf.UserId == userId || uf.FriendId == userId) && uf.Status == FriendshipStatus.Accepted)
    .Select(uf => uf.UserId == userId ? uf.FriendId : uf.UserId)
    .Distinct()
    .ToListAsync();

var allUserIds = userReadMap
    .Select(rl => rl.UserId)
    .Distinct()
    .Where(uid => uid != userId && !existingFriendships.Contains(uid))
    .ToList();



var predictionEngine = mlContext.Model.CreatePredictionEngine<UserBookEntry, FriendPrediction>(model!);

var scores = new List<(int otherUserId, float score)>();
```

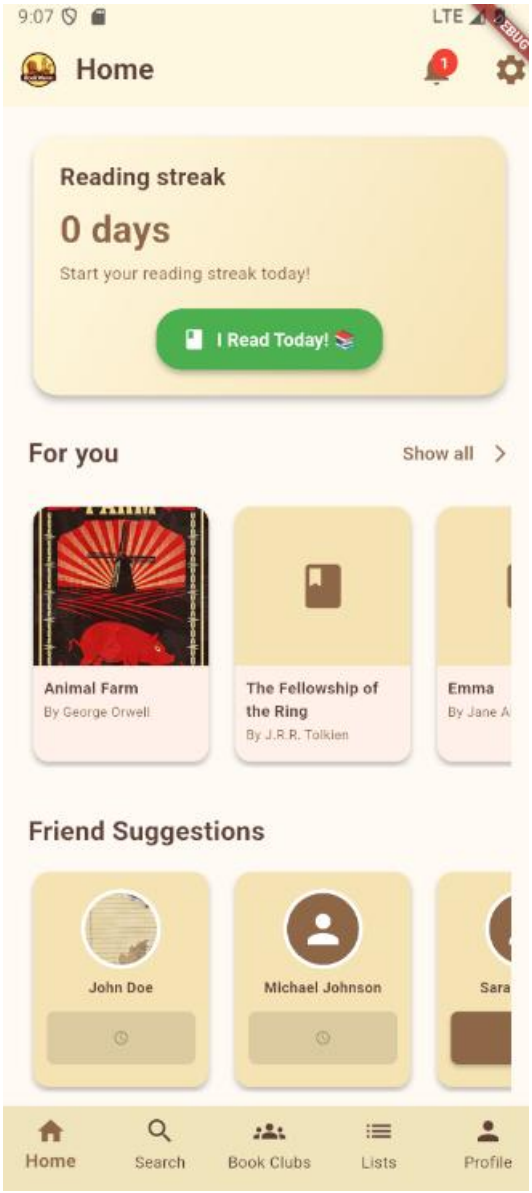
```
    foreach (var otherUserId in allUserIds)
    {
        var score = 0f;

        var sharedBooks = userReadMap
            .Where(rl => rl.UserId == userId)
            .SelectMany(rl => rl.ReadingListBooks.Select(rb => rb.BookId))
            .Distinct()
            .ToList();

        foreach (var bookId in sharedBooks)
        {
            var prediction = predictionEngine.Predict(new UserBookEntry
            {
                UserId = (uint)userId,
                BookId = (uint)bookId
            });

            score += prediction.Score;
        }

        scores.Add((otherUserId, score));
    }

    var topUserIds = scores
        .OrderByDescending(x => x.score)
        .Select(x => x.otherUserId)
        .Distinct()
        .Take(10)
        .ToList();

    if (!topUserIds.Any())
    {
        return await GetMostActiveUsersFallback(userId);
    }

    var users = await _context.Users
        .Where(u => topUserIds.Contains(u.Id))
        .ToListAsync();

    return _mapper.Map<List<UserResponse>>(users);
}
```

Putanja do code-a u aplikaciji gdje se poziva recommender sistem:

BookWorm\BookWorm\UI\bookworm_mobile\lib\screens\homepage.dart

Preporuke prijatelja se također prikazuju na početnoj stranici, odmah ispod predloženih knjiga. Ako je model već istreniran, preporuke se odmah učitavaju iz fajla (ukolko je treniranje rađeno taj isti dan), a ukoliko nije ponovno se trenira model.

Printscreen iz pokrenute aplikacije


9:07
LTE
DEBUG
Home
1
Reading streak
0 days
Start your reading streak today!
I Read Today!
For you
Show all
Animal Farm
By George Orwell
The Fellowship of the Ring
By J.R.R. Tolkien
Emma
Friend Suggestions
John Doe
Michael Johnson
Sara
Home
Search
Book Clubs
Lists
Profile