TP1 Sistemas de Recomendação

Pablo Goulart Silva

Introdução

A filtragem colaborativa permite escolher grupos de usuários ou itens que compartilham características semelhantes para prover boas recomendações de maneira automática. Nesse trabalho é apresentado a implementação de uma técnica orientada a itens que permite quantificar o interesse do usuário por um item baseado na avaliação dos usuários por itens correlacionados.

A matriz usuário-item apresenta a avaliação explícita dos usuários nos itens existentes. Um subconjunto de itens é selecionado de acordo com sua similaridade com o item sob avaliação e a soma ponderada das avaliações dos similares são utilizadas para gerar predições. O algoritmo de filtragem colaborativa baseada em vizinhanças pode ser dividido em três passos:

- cálculo da similaridade entre todos os itens
- seleção de um subconjunto de itens similares para serem preditores
- normalização e combinação ponderada das avaliações dos itens mais similares.

O número de itens co-avaliados foi considerado e a hipótese de melhoria da acurácia do algoritmo para variações no tamanho do conjunto de itens foi validada nesse trabalho. Para determinar os itens utilizados como preditores do item ativo visando cobertura, acurácia e factibilidade, os vizinhos foram selecionados utilizando a técnica *melhores-n-vizinhos*.

Representação

A matriz usuário-item foi armazenada como um dicionário de dicionários (1). A chave do dicionário representa o identificador do usuário. O dicionário aninhado representa o identificador do item e sua avaliação. A busca `dicionário`[`usuarioId`] devolve um dicionário de itens que pode ser indexado pelo identificador do item. `dicionário`[`usuarioId`][`itemId`] devolve a estrutura `ItemPrediction` que possui os valores `timestamp` e `prediction` da avaliação do usuário sobre o item (3).

Para a representação orientada a itens, utilizamos o dicionário análogo para representar a avaliação de cada item pelos usuários (2). Cada item representa uma chave no dicionário cujo valor é um dicionário de usuários.

```
{
  "usuario_1":
  {
    "item_1":
    {
        rating_{11},
        timestamp
    },
    "item_2":
    {
        rating_{12},
        timestamp
    }
  },
  "usuario_2":
  {
    "item_1":
    {
        rating_{21},
        timestamp
    },
    "item_2": {
        rating_{22},
        timestamp
    }
  }
}
```

Listing 1: Representação da estrutura de dados para armazenar a matriz usuário-item.

A representação em dicionários foi escolhida para devido a esparsidade da matriz usuário-item, além de possuir complexidade de pesquisa de $logN$, onde N é o número de usuários para a matriz usuário-item, e $logM$ para a matriz item-usuário, onde M é o número de itens.

Algoritmo

Nesse trabalho foram implementadas as abordagens orientadas a usuário e orientada a itens. A primeira abordagem prediz a avaliação do usuário sobre um item a partir da avaliação desse item pelos seus usuários mais similares, enquanto a segunda abordagem calcula a predição a partir da avaliação do usuário sobre itens semelhantes ao item alvo. As avaliações presentes no conjunto de treina-

```
{
  "item_1":
  {
    "usuario_1":
    {
        rating_{11},
        timestamp
    },
    "usuario_2":
    {
        rating_{12},
        timestamp
    }
  },
  "item_2":
  {
    "usuario_1":
    {
        rating_{21},
        timestamp
    }
    "usuario_2":
    {
        rating_{22},
        timestamp
    }
  }
}
```

Listing 2: Representação da estrutura de dados para armazenar a matriz item-usuário.

```
struct ItemPrediction {
    time_t timestamp;
    int prediction;
};
```

Listing 3: Estrutura de dados representando a avaliação de um item pelo usuário.

mento seguem uma distribuição de *Weibull*. O gráfico da figura 2 foi ajustado a partir de uma amostragem dos dados da base (trechos de código 6 e 7).

```
typedef map<string, map<string, ItemPrediction> > RatingMatrix;
RatingMatrix [item,user]BasedMatrix;
```

Listing 4: Estrutura de dados para armazenar as avaliações dos usuários par um item.

```
typedef map<string, map<string, double> > SimilarityType;
SimilarityType similarity;
```

Listing 5: Estrutura de dados para armazenar a similaridade entre itens/usuários.

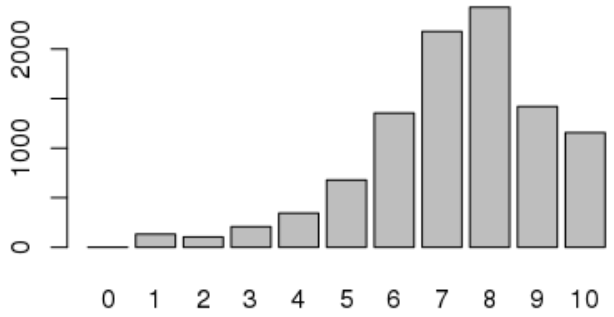

Figura 1: Distribuição das avaliações de usuários sobre itens da base de treinamento.

```
data <- read.csv(file = 'ratings.csv', header = TRUE, sep = ',')
num.items <- 10000
sampling <- sample(length(data$Prediction))[1:num.items]
samples <- data$Prediction[sampling]
barplot(table(samples))
```

Listing 6: Código para cálculo da distribuição das avaliações dos dados de treinamento em *R*.

Recomendação orientada ao usuário

Implementado pelo método `UserToUser`, o algoritmo prediz a avaliação de um usuário sobre um item a partir das avaliações de usuários similares para o mesmo

```
library(MASS)
fit.w <- fitdistr(samples, "weibull")
shape <- fit.w$estimate[1]
scale <- fit.w$estimate[2]
qqplot(samples, rweibull(1000, shape, scale))
cat("Shape: ", shape)
cat("Scale: ", scale)

> Shape:  4.613627
> Scale:  7.947462
```

Listing 7: Cálculo do ajuste de distribuição a partir dos dados de treinamento em *R*.

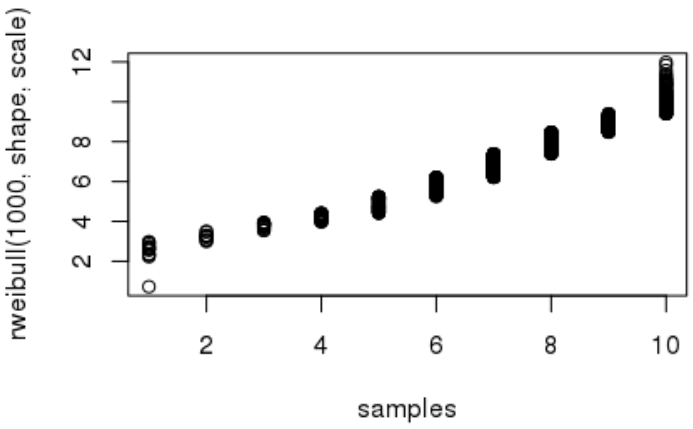

Figura 2: Gráfico *quantile-quantile* do ajuste dos dados da base de treinamento à uma distribuição de Weibull com parâmetros estimados.

item. Para cada item da base de teste, o algoritmo:

- calcula os melhores-N-vizinhos (ou seja, os N usuários mais similares) do usuário atual
- calcula a média das avaliações do usuário-alvo
- calcula a soma ponderada das avaliações dos usuários similares sobre o item-alvo
- se a soma ponderada for zero (ou denominador for nulo), amostrar a avaliação do usuário-alvo sobre o item a partir da distribuição de *Weibull*.

O método `TopMatches()` calcula os melhores N vizinhos para o usuário a partir da matriz *userBasedMatrix*, e os melhores N vizinhos para os itens a partir da matriz *itemBasedMatrix* (definição 8). Esse método recebe como parâmetro as matrizes usuário-item e item-usuário, o identificador do usuário/item alvo, uma função de similaridade que implemente a interface definida no trecho 9, valores P e K indicando, respectivamente, tamanho do subconjunto de similares amostrado da base, e tamanho do subconjunto de similares escolhidos. O parâmetro *randomize* define se a amostragem é aleatória ou sequencial.

```
struct TopMatches {
    vector<pair<string, double> > operator()(
        map<string, map<string, ItemPrediction> >&,
        map<string, map<string, ItemPrediction> >&,
        string,
        function<double(
            map<string, map<string, ItemPrediction> >&,
            map<string, map<string, ItemPrediction> >&,
            const string&,
            const string&)>,
        int K=-1, int P=-1, bool randomize=false
    );
};
```

Listing 8: Definição método *TopMatches*.

```
struct BaseSimilarity {
    double operator()(
        map<string, map<string, ItemPrediction> >&,
        map<string, map<string, ItemPrediction> >&,
        const string&,
        const string&);
};
```

Listing 9: Interface dos métodos de similaridade.

O algoritmo verifica se existem itens similares ao item sob avaliação na matriz item-usuário. Itens similares são avaliados pela ponderação dos itens existentes. Caso não haja itens similares, verifica-se as avaliações anteriores para o usuário. A avaliação predita é o resultado da média das avaliações passadas. Em caso de novo usuário, é retornado uma amostra de distribuição de *Weibull*.

A recomendação orientada a usuários possui complexidade $O(N(N+kNlogN))$, onde N é o número de itens, sendo que $N + kNlogM$ corresponde a chamada do método `TopMatches` que calcula a similaridade entre N usuários e os ordena em $O(NklogN)$, para $k \leq 1$ representando a fração dos usuários comparados.

```
UserToUser() {
    RatingMatrix userBasedMatrix, itemBasedMatrix
    SimilarityType similarity

    foreach target in Targets:
        foreach item in similarity[target.itemId]:
            num = num + userBasedMatrix[target.userId][item.Id];
            den = den + item.similarity;
        if den != 0:
            print 'similarity: ' + num/den
        else:
            if len(userBasedMatrix[target.userId]) > 0:
                print 'similarity: ' + UserAverageScore(target.userId)
            else:
                print 'similarity: ' + weibull_generator()
```

Listing 10: Pseudo-código do método de recomendação orientado a usuários.

```
ItemToItem() {
    RatingMatrix userBasedMatrix, itemBasedMatrix
    SimilarityType similarity

    foreach target in Targets:
        foreach item in similarity[target.itemId]:
            num = num + userBasedMatrix[target.userId][item.Id];
            den = den + item.similarity;
        if den != 0:
            print 'similarity: ' + num/den
        else:
            if len(userBasedMatrix[target.userId]) > 0:
                print 'similarity: ' + UserAverageScore(target.userId)
            else:
                print 'similarity: ' + weibull_generator()
```

Listing 11: Interface método de similaridade.

Recomendação orientada a itens

A recomendação orientada a itens possui complexidade dominada pelo método `calcularSimilares()`, que é $O(M(M + kMlogM))$, onde M é o número de itens, sendo que $M + MlogM$ corresponde a chamada do método `TopMatches` que calcula similares para M itens e os ordena em $O(kMlogM)$ e $k \leq 1$ representa a fração dos itens comparados. A complexidade da predição possui complexidade $O(VcM)$, onde V é o número de itens alvos, e cM representa o subconjunto de itens similares selecionado por `TopMatches`, para $c \leq 1$.

O algoritmo mantém dois dicionários durante a execução: *userBasedMatrix* e *itemBasedMatrix*. O método `createItemBasedMatrixFromUserBasedMatrix` inverte a representação exibida no trecho 1 para criar a matriz item-usuário (trecho 2) armazenada pela variável *itemBasedMatrix*.

O algoritmo de recomendação orientado a itens requer o cálculo antecipado da matriz de similaridade entre itens. Essa matriz é armazenada como um dicionário em que cada elemento é outro dicionário cujo valor é a avaliação do item, conforme exibido no trecho de código 12.

```
typedef map<string, map<string, double> > SimilarityMatrix;
```

Listing 12: Definição do tipo matriz de similaridade.

Após calculada a matriz de similaridades, o algoritmo itera sobre os usuários/itens sob avaliação e calcula a predição da avaliação utilizando as regras abaixo:

- calcular a similaridade de um item que possua itens semelhantes
- calcular a média das avaliações do usuário atual que possua itens avaliados
- amostrar a avaliação de uma distribuição de *Weibull* ajustada a partir da distribuição das avaliações do conjunto de treinamento.

Resultados

O algoritmo de recomendação orientado ao usuário foi avaliado utilizando a correlação de *Pearson*. A correlação de Pearson apresenta resultados empíricos superiores para recomendação orientada a usuários [1]. Os valores K e P foram definidos nos intervalos $[20, 50]$, e $[50, 100]$, respectivamente, para amostragem e cálculo de usuários similares. Essa abordagem apresentou valores de $RMSE$ superiores a recomendação orientada a itens.

O algoritmo de recomendação orientado a itens foi escolhido para a versão final. De acordo com [1], a similaridade *cosseno* é superior para comparação de itens. Foram utilizadas as seguintes parametrizações ao longo dos experimentos:

Tabela 1: Resultados do algoritmo orientado a itens.

Correlação	K	P	Randomize	Distribuição	Média/Usuário	Tempo	RMSE
Pearson	-1	-1	Não	-	Não	>5	-
Cosseno	-1	-1	Não	-	Não	>5	-
Pearson	40	60	Sim	Uniforme	Não	<5	4.00652
Cosseno	40	60	Sim	Uniforme	Não	<5	3.69342
Cosseno	40	60	Sim	Weibull	Não	$\approx 4m50s$	2.59595
Cosseno	40	60	Sim	Weibull	Sim	$\approx 4m40s$	1.99518

- correlação de *Pearson*, *distância Euclidiana*, *cosseno* e *cosseno ajustado*

- busca de itens/usuários em todo o espaço amostral (valores de K e P iguais a -1)
- valores de K e P definidos nos intervalos $[20, 50]$ e $[50, 100]$, respectivamente
- seleção de itens aleatória (uniforme) e sequencial
- amostragem de avaliação dos usuários sobre os itens em distribuição uniforme e *Weibull*.

A tabela 1 apresenta os resultados obtidos para as parametrizações utilizadas no trabalho. Os resultados omitidos apresentaram tempo de execução superior a cinco minutos e/ou $RMSE > 3$.

A utilização da distribuição de *Weibull* diminuiu o $RMSE$ dos casos onde itens novos eram avaliados por usuários sem nenhuma avaliação anterior. Nesses casos, o algoritmo estimou um valor de avaliação próximo a distribuição da base. Os usuários tendem a concentrar suas avaliações entre 5 e 10, de acordo com o gráfico 2.

Outro fator de melhoria foi a utilização da média das avaliações do usuário para os casos de usuários com histórico avaliando itens novos, denotando a consistência dos usuários para avaliar itens com valores próximos às suas avaliações anteriores.

Referências

[1] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.