cf2vec: Collaborative Filtering algorithm selection using graph distributed representations

Juan Valdivia

Advanced Analytics Prod

juan.valdivia@itau-unibanco.com.br

11/10/2021

Agenda

1. Introdução
   - Motivação
2. *Sistemas de recomendação*
   - *Collaborative filtering* (*CF*)
3. *Metalearning*
   - Recomendação de algoritmos usando *Metalearning*
4. Representational Learning
5. *cf2vec*
6. *Resultados*

Introdução

- **Desafio:** Falta de orientação sobre qual algoritmo de *Collaborative Filtering* (abordagem de sistemas de recomendação) seria mais adequado para uma determinada tarefa (por exemplo: *Rating Prediction*).
- **Solução 1:** Selecionar algoritmos para uma determinada tarefa pode ser um processo experimental árduo (abordagem de tentativa e erro), custoso, subjetivo, podendo levar ao *overfitting* e dificulta a reprodução de experimentos.
- **Solução 2:** abordagem de *Metalearning* (*MtL*) que consiste basicamente em extrair metaconhecimento de tarefas resolvidas com sucesso por *ML* para usá-lo na solução de novas tarefas.

Objetivo

O objetivo deste trabalho é projetar automaticamente *metafeatures* para o problema de seleção de algoritmos em filtragem colaborativa usando outros modelos de *embeddings* para grafos como por exemplo *node2vec*, *struct2vec* e *Anonymous Walk Embeddings*.

Collaborative filtering (CF)

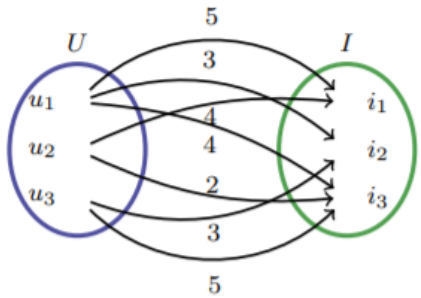
- Abordagem de sistemas de recomendação que consiste em filtrar informações ou padrões usando técnicas que envolvem colaboração entre vários usuários, agentes e fontes de dados.
- *CF* usa a interação dos usuários com os itens por meio de um *feedback*.
- *Feedback* explícito: usuários tomam a iniciativa de indicar seus interesses.
  - Problema: usuários podem não ter interesse em avaliar os itens
- *Feedback* ímplicito: presente no comportamento dos usuários nos serviços online (histórico de compras, histórico de navegação e cliques)
  - Problema: comportamento do usuário pode não refletir necessariamente de maneira direta a sua opinião sobre o produto.

Collaborative filtering: *Como são os conjuntos de dados?*

- Para o caso do *feedback* explícito, podemos representar o conjunto de dados como um matriz de *ratings* ou como um grafo bipartido.

	i_1	i_2	i_3
u_1	**5**	**3**	**4**
u_2	**4**	...	**2**
u_3	...	**3**	**5**

(a) Rating Matrix. Rows represent users U, while columns represent items I. Some cells have the rating assigned by an user to an item.



(b) Bipartite Graph. The graph has two node subsets, representing users U and items I. Ratings are weighted edges between nodes of both subsets.

Figure: Representações de um conjunto de dados de filtragem colaborativa

Collaborative filtering: Tarefas

Podem ser diferenciadas com base nos tipos de *feedback* e dados de entrada:

- *Rating Prediction*: previsão das classificações explícitas (*rating*) dadas pelos usuários aos itens. A métrica *RMSE* pode ser usada.
- *Item Recommendation*: recomendação de uma lista de itens que é mais adequada para um determinado usuário. A métrica *AUC* pode ser usada.

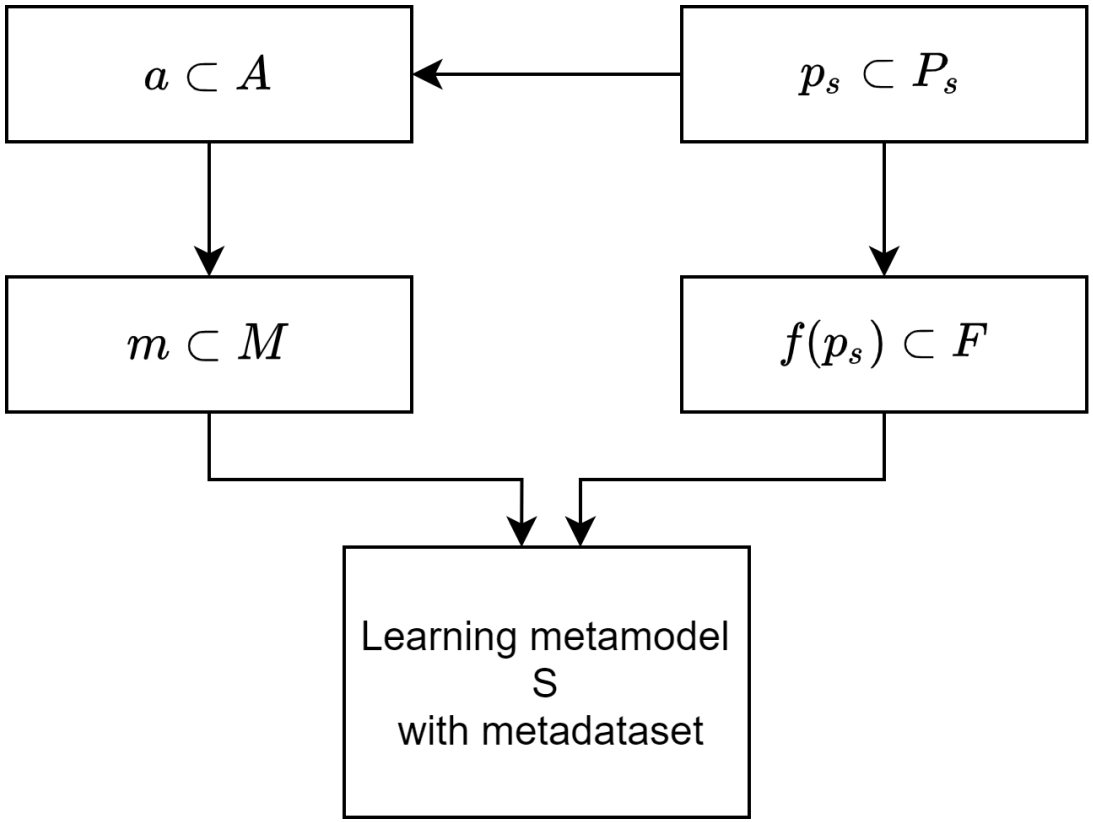
Metalearning x *Machine Learning*

- *MtL* possui dois níveis (*levels*) de adaptação: o *baselevel* e o *metalevel*.
- No *baselevel* o aprendizado foca em acumular experiência em um tarefa específica.
  - No aprendizado supervisionado, um *base-learner* seria um classificador tradicional.
- No *metalevel* o aprendizado foca em acumular experiência através do desempenho de múltiplas aplicações de um sistema de aprendizado.
  - Um *meta-learner* mapeia as *metafeatures* em *metatargets*. Esse *meta-learner* é um algoritmo tradicional de aprendizado de máquina adaptado para a tarefa que o sistema de *MtL* precisa resolver.

Metalearning: O que é necessário para selecionar algoritmos?

- Conjunto de diversos *datasets* p_s que pertencem a um problema P_s.
- *Metafeatures* (F): extraem propriedades dos *datasets* p_s na forma $f(p_s)$. Essas propriedades devem prover evidência sobre o desempenho futuro das técnicas investigadas, devem distinguir problemas com diferentes níveis de dificuldade e preferencialmente devem ser calculadas com um baixo custo computacional.
- Conjunto de algoritmos A: cada algoritmo $a \subset A$ é aplicado a cada conjunto de dados p_s, ou seja tem-se, $a(p_s)$.
- Métrica de avaliação: para cada métrica $m \subset M$ é feita a avaliação dos modelos obtidos após a aplicação de cada um dos algoritmos a nos conjuntos de dados p_s, ou seja, $m(a(p_s))$.

Metalearning: Arcabouço de seleção de algoritmos



1) Selecionar um repositório de conjuntos de dados P_s
2) Usar cada um dos A algoritmos em cada um dos conjuntos de dados de P_s
3) Medir o desempenho usando uma métrica M para cada um dos algoritmos de A em cada conjunto de dados de P_s
4) Extrair as metafeatures de cada um dos conjuntos de dados de P_s

Figure: Arcabouço de seleção de algoritmos

Principais *metafeatures* para *datasets* de *collaborative filtering*

- *Rating Matrix metafeatures*: estatísticas (min, max, moda, curtose, entropia) sobre as linhas/colunas, número de usuários, estatísticas sobre os *ratings*.
- *Subsampling landmarkers*: extraídas usando o desempenho estimado para amostras aleatórias dos *datasets* originais.
- *Graph metafeatures*: os *datasets* de *CF* são modelados como um grafo e as características extraídas são medidas usadas em redes complexas.
- *Comprehensive metafeatures*: todas as abordagens anteriores para extrair *metafeatures* são agregadas. Para obter as *metafeatures* mais significativas é usada seleção de *features* por correlação.

Como seria esse sistema de recomendação de algoritmos?

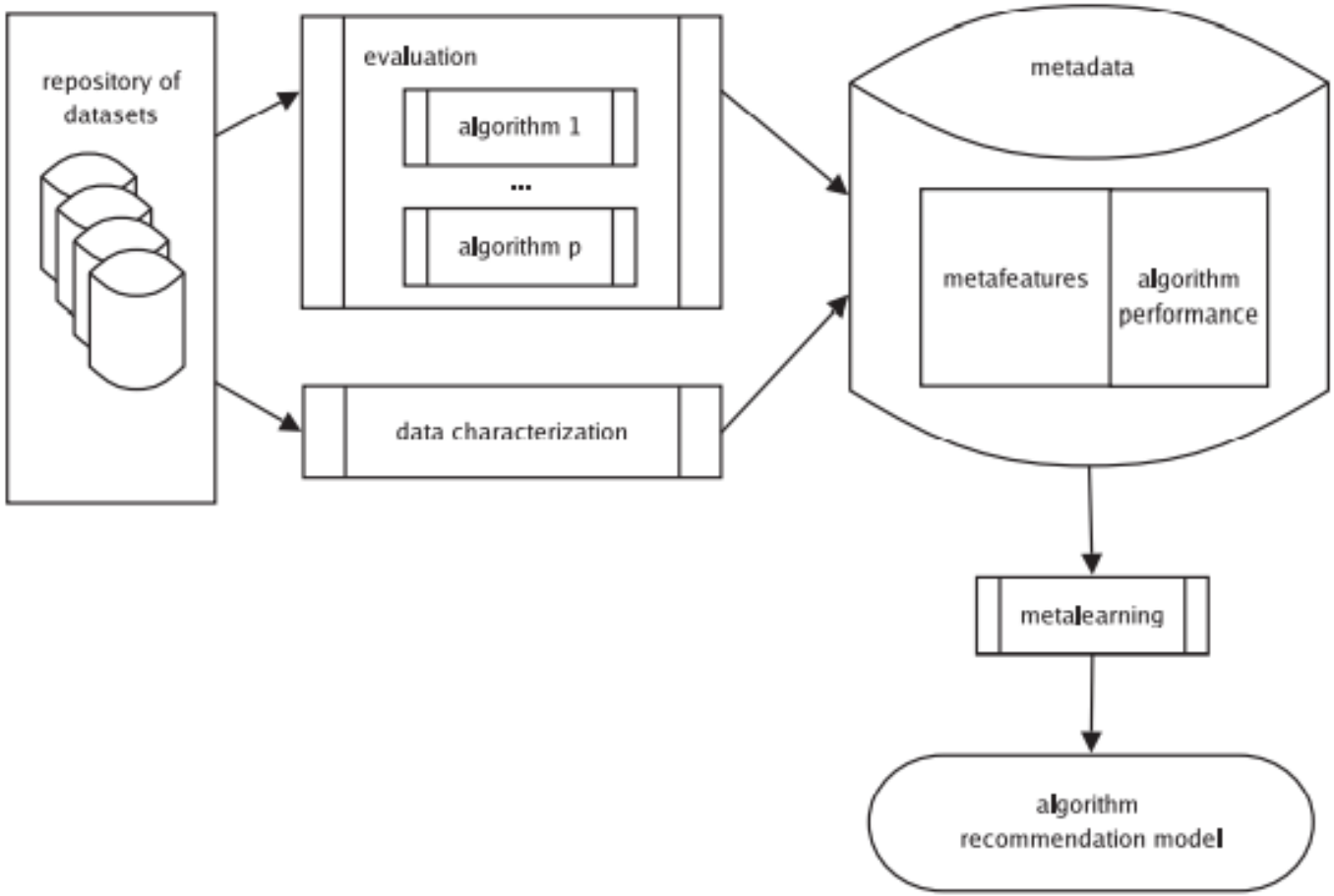


Figure: Sistema de recomendação de algoritmos usando *Metalearning*

k-Nearest Neighbors Ranking Method

Para treinar *metadatasets* é necessário um *meta-learner*. O *k-Nearest Neighbors Ranking Method* é uma adaptação do *k-Nearest Neighbors* para a tarefa de fazer um *ranking* de algoritmos.

1. Selecionar o conjunto de k exemplos mais similares para um novo exemplo (no caso um *dataset* em *MtL*) com relação às *metafeatures*. Para isso deve ser usada uma métrica de distância para medir a similaridade.
2. Combinar os valores dos *targets* dos k exemplos selecionados para gerar uma previsão para o novo exemplo (*dataset*).

Node Representational Learning

- Objetivo: mapear a estrutura topológica complexa de um grafo em um espaço dimensional menor no qual a similaridade entre os *embeddings* dos nós do grafo indique similaridade entre grafos.

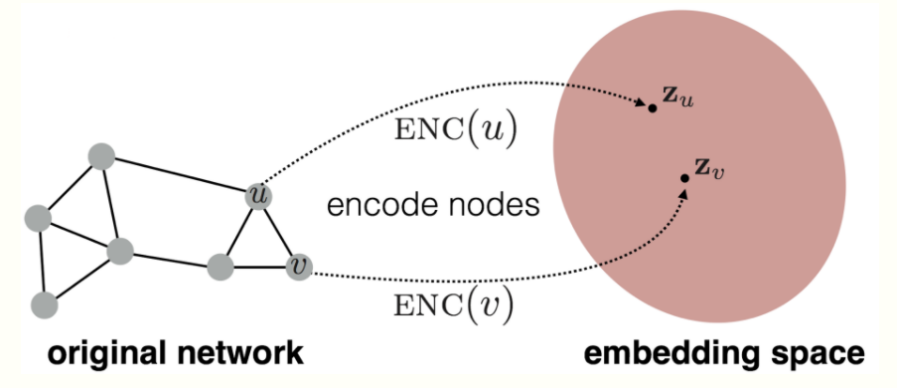
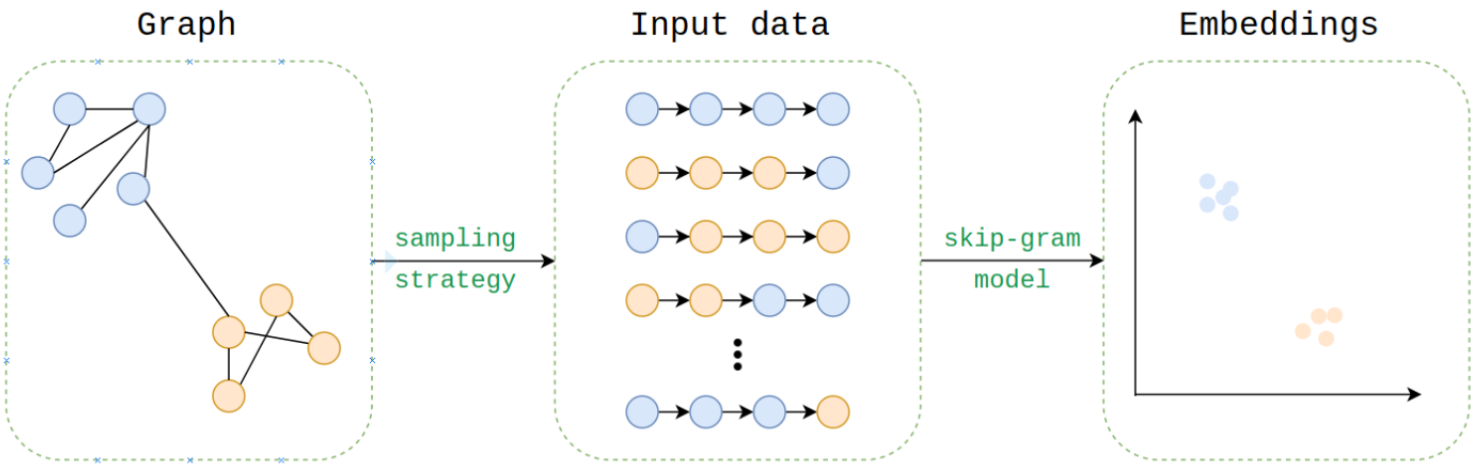


Figure: Processo de mapeamento do espaço de nós para o espaço de *embeddings*

Node Representational Learning: *node2vec*

- Usa o modelo *skip-gram* para encontrar *embeddings* para nós.
- Ideia: utilizar várias caminhadas aleatórias enviesadas de segunda ordem que podem fazer o balanceamento entre visões locais e globais da rede.
- *Random walks* oferecem uma definição estocástica flexível de similaridade entre nós que incorpora informação das vizinhanças locais e globais.
  - Vantagem: não é preciso treinar todos os pares de nós, apenas consideraramos pares que ocorrem dentro de uma mesma caminhada aleatória.



Node Representational Learning: *struc2vec*

- Proposta por brasileiros (UFRJ).
- Ideia: aprender *embeddings* que capturem a identidade estrutural (relacionado com o papel do nó dentro do grafo) dos nós.
  - Resultado: a distância entre os *embeddings* dos nós está altamente correlacionada com a similaridade estrutural entre eles.

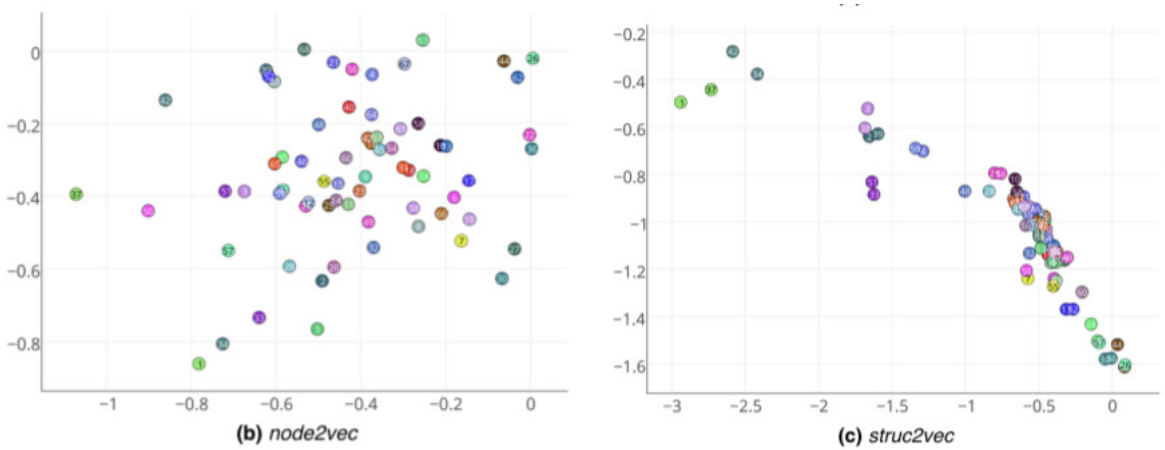


Figure: Comparação dos *embeddings* gerados pelo *node2vec* e *struc2vec* para o grafo do *Zachary's Karate Club*

Como podemos encontrar *embeddings* para o grafo todo?

- Ideia simples: encontrar os *embeddings* de cada um dos nós do grafo G usando alguma técnica de *node representational learning*. Logo depois basta só somar (ou calcular a média) dos *embeddings*.
  - $z_G = \sum_{v \in G} z_v$ (no caso da soma) ou $z_G = \frac{1}{|v|} \sum_{v \in G} z_G$ (no caso da média).

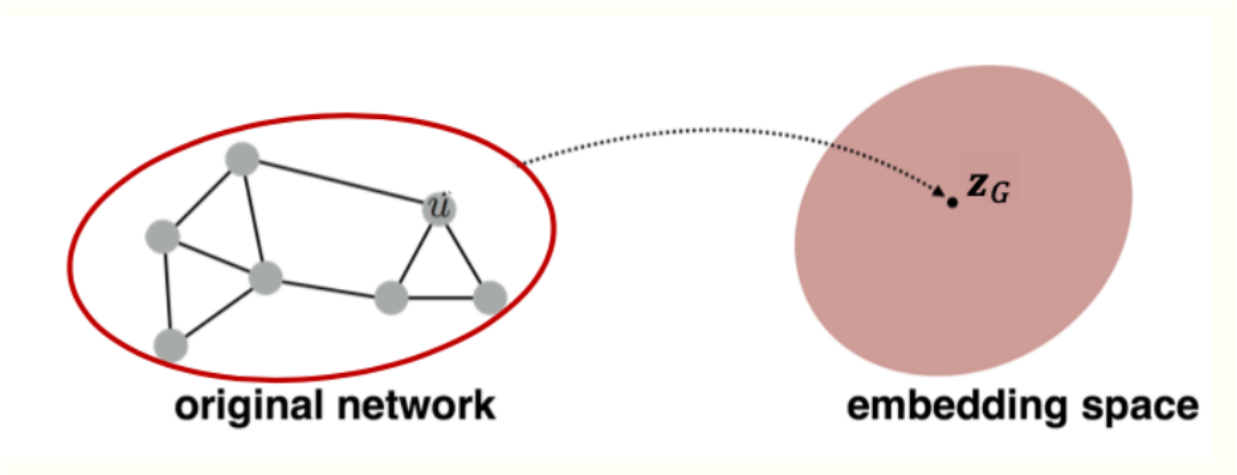


Figure: Processo de mapeamento do espaço de grafos para o espaço de *embeddings*

Graph Representational Learning (continuação)

Como podemos encontrar *embeddings* para o grafo todo?

- *Anonymous Walk Embeddings* enumera todas as possíveis caminhadas anônimas de l passos, registra as contagens e representa o grafo como uma distribuição de probabilidades sobre essas caminhadas.
- Usar o *graph2vec*: adaptação do *doc2vec* (aprende representações distribuídas para sequências de palavras de diferentes tamanhos como por exemplo parágrafos e documentos).

Como o sistema de recomendação de algoritmos do *cf2vec* funciona?

- Converte a matriz de filtragem colaborativa em um grafo bipartido.
- Amostra o grafo bipartido para reduzir a complexidade do problema.
- Aprende *embeddings* (representações distribuídas para cada um dos grafos) usando *graph2vec*.
- Utiliza os *embeddings* aprendidos para cada um dos grafos como *metafeatures* do *metadataset* que será construído.

Framework do *cf2vec*: Extração de *metafeatures*

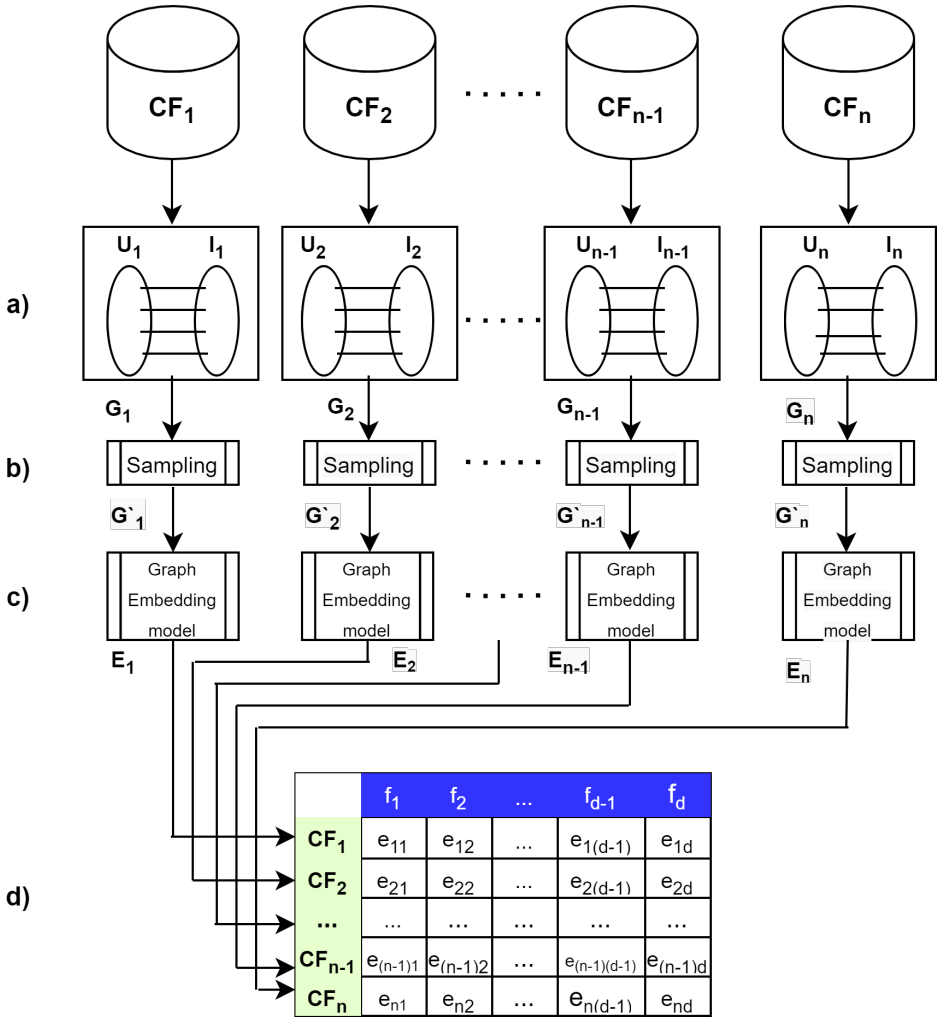
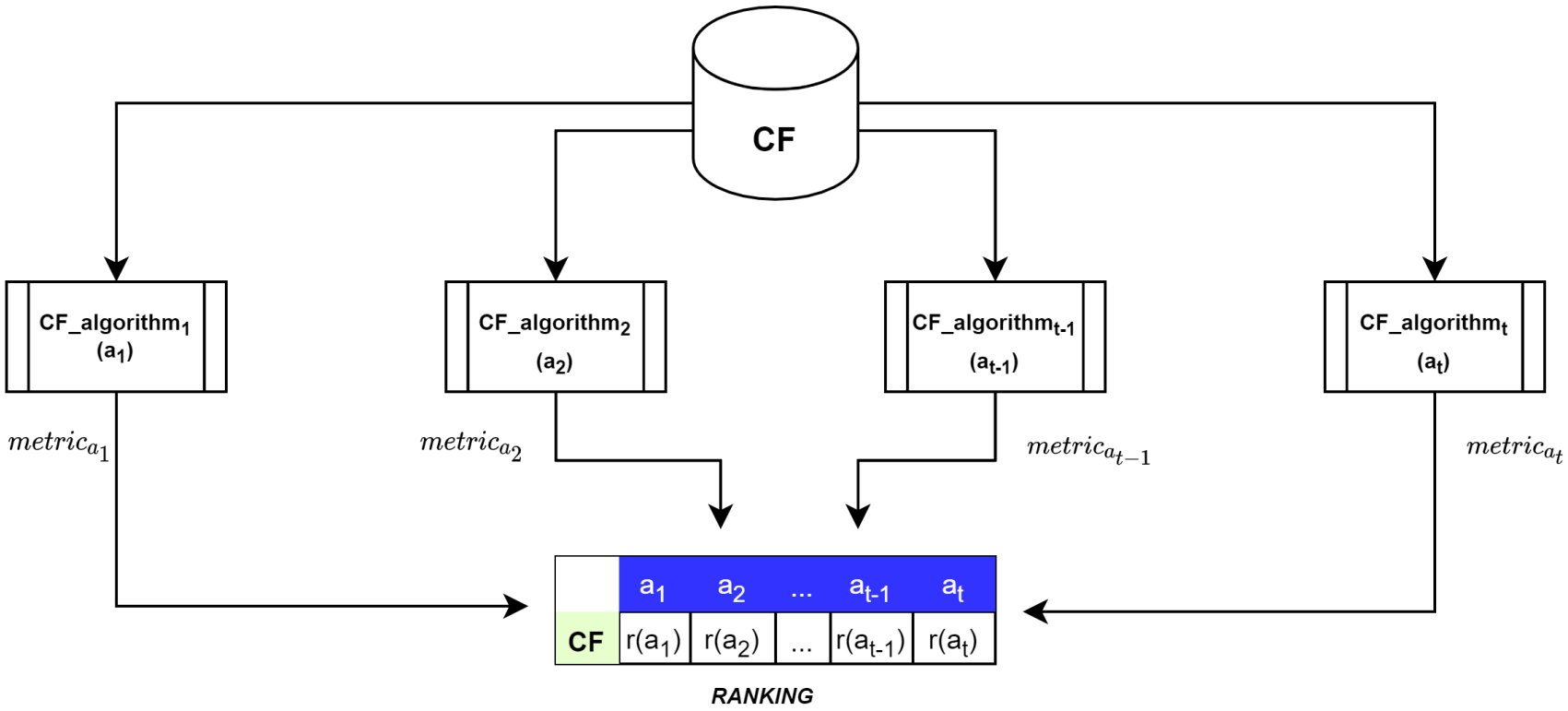


Figure: Extração de *metafeatures* usando o *framework cf2vec*

Framework do *cf2vec*: Criação dos *metatargets*

Para cada conjunto de dados aplica-se t algoritmos de *CF* e avalia-se usando uma métrica específica para cada tarefa que foi realizada. Dessa forma, a partir da métrica é possível gerar um *ranking* para poder recomendar os algoritmos.



Metamodelo

Metamodelos são avaliados usando a métrica *Kendall's Tau* (coeficiente que representa o grau de concordância entre duas colunas de *rankings*). Essa métrica é calculada usando a Equação 1 na qual C é o número de pares concordantes e D é o número de pares desconcordantes.

$$\tau = \frac{C - D}{C + D} \tag{1}$$

Framework do *cf2vec*: Metamodelo

- Avalia-se o metamodelo usando *leave one out cross-validation*.
- hiperparâmetros são ajustados usando *grid search*.

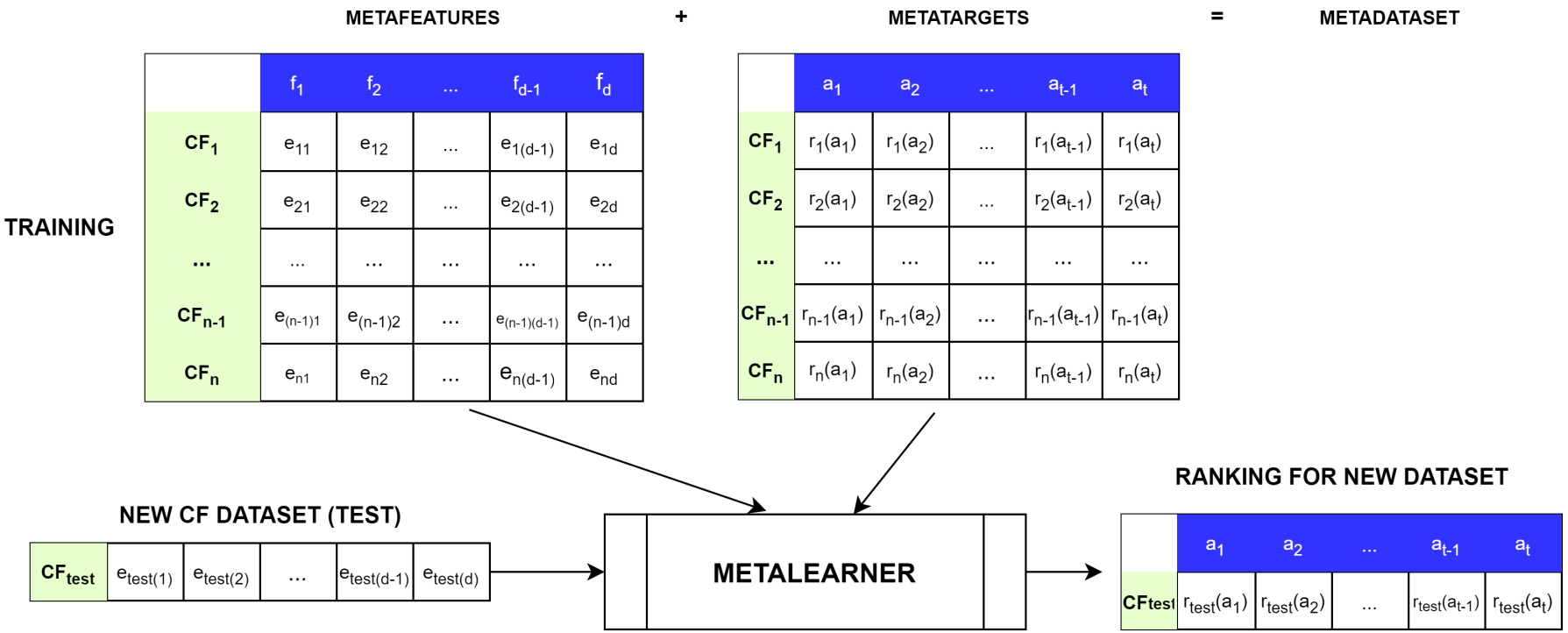


Figure: *Metamodel* do *cf2vec*

Experimentos: Setup experimental

- O metalearner *k-Nearest Neighbors Ranking Method* foi executado para 38 *datasets*.
  - *Amazon* (22 *datasets*), *Bookcrossing* (1 *dataset*), *Flixter* (1 *dataset*), *Jester* (3 *datasets*), *MovieLens* (5 *datasets*), *MovieTweetings* (2 *datasets*), *TripAdvisor* (1 *dataset*), *Yahoo*! (2 *datasets*) e *Yelp* (1 *dataset*).
- A média dos *embeddings* dos nós do grafo extraídos por *node2vec* e *struct2vec* foram usados para extrair as *metafeatures* dos *datasets*.
- *graph2vec* e *AWE* foram usados também para extrair as *metafeatures*.

Experimentos: Setup experimental (continuação)

Duas tarefas:

- *Rating Prediction* (*RP*)
  - algoritmos: *Matrix Factorization (MF)*, *Biased MF (BMF)*, *Latent Feature Log Linear Model (LFLLM)*, *SVD++*, 3 versões de *Sigmoid Asymmetric Factor Model* (*SIAFM*, *SUAFM* e *SCAFM*), *User Item Baseline (UIB)* e *Global Average (GA)*.
  - métricas: *NMAE* e *RMSE*
- *Item Recommendation*
  - algoritmos: *BPRMF*, *Weighted BPRMF (WBPRMF)*, *Soft Margin Ranking MF (SMRMF)*, *WRMF* e *Most Popular (MP)*.
  - métricas: *NDCG* e *AUC*

Diagrama de Diferença Crítica

- Usado para comparar o desempenho de múltiplos algoritmos em múltiplos conjuntos de dados.
- Múltiplos testes de hipóteses são conduzidos sobre a saída (desempenho) dos algoritmos usando o teste de Friedman para dizer se a diferença estatística significante entre os algoritmos.
- A posição representa a média do ranking para todas as saídas de todos os datasets.

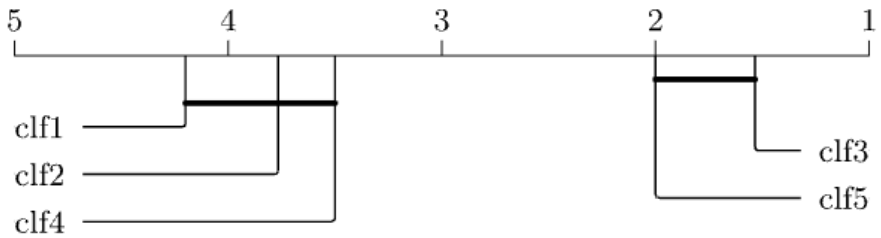
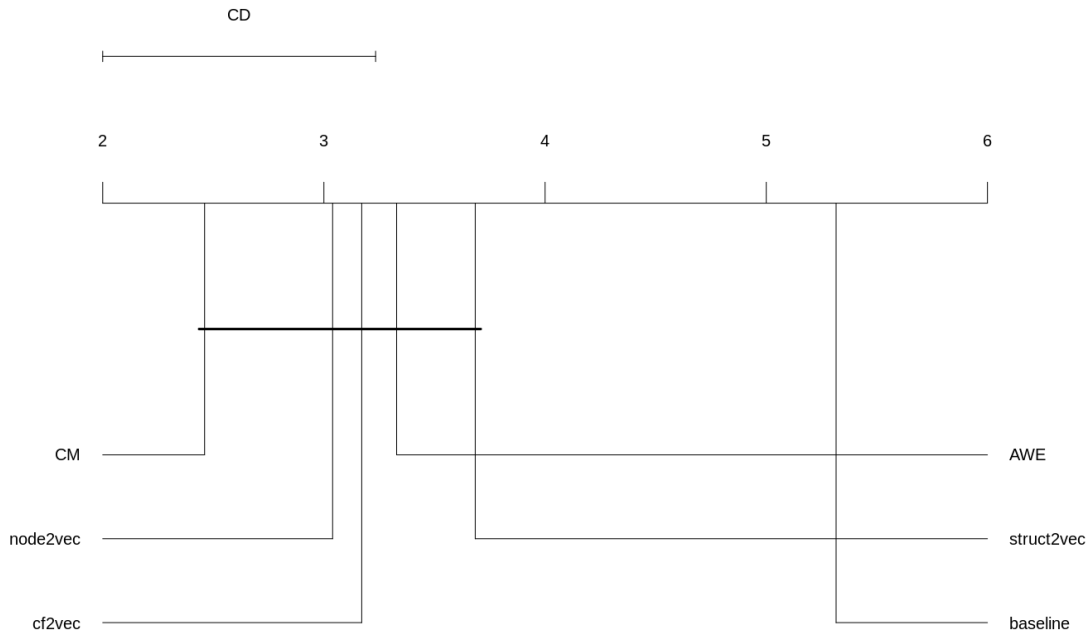


Figure: Exemplo de diagrama de Diferença Crítica

Experimentos: Diagrama de Diferença Crítica

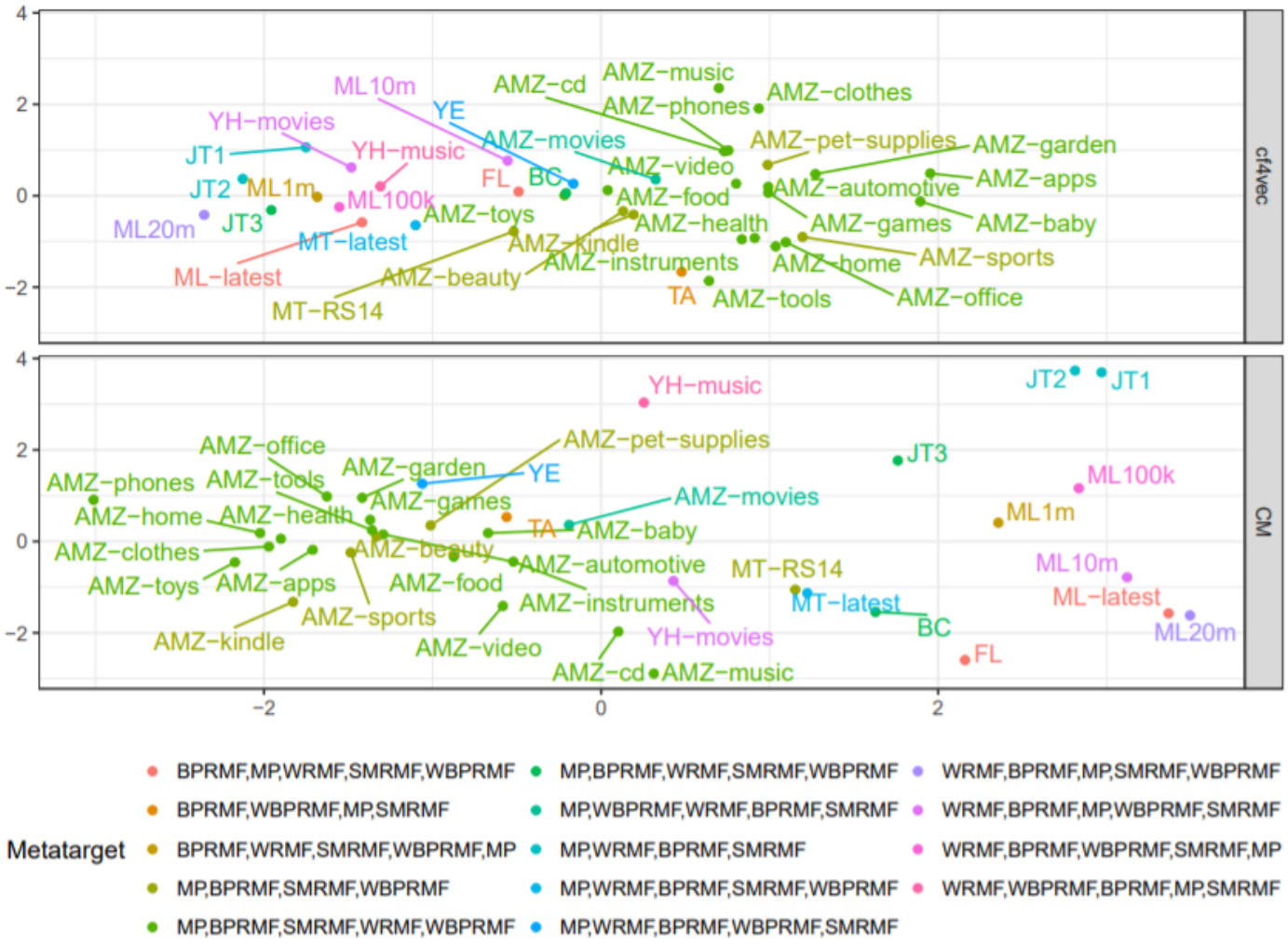
- Diagrama de Diferença Crítica: cada estratégia para extração de *metafeatures* é representada pelo melhor desempenho do metamodelo para todos os conjuntos de dados utilizados.
- Não há diferença estatisticamente significante entre as *Comprehensive metafeatures* e as *metafeatures* extraídas pelas técnicas de *embeddings* utilizadas.



Experimentos: Entendendo o relacionamento entre as *metafeatures* extraídas e os *metatargets* usando PCA

- Duas componentes do *PCA* projetadas para visualizar esses resultados de modo que o *ranking* dos *baselearners* para cada conjunto de dados é mostrado usando uma gama de cores baseada na similaridade dos *metatargets*.
  - Conjuntos de dados com *embeddings* similares estão próximos nessa projeção.
  - Se o *ranking* de algoritmos for similar para dois conjuntos similares as cores serão similares.

Experimentos: Resultados do PCA para *Item Recommendation*



Experimentos: Intepretação dos resultados do PCA

As *metafeatures* funcionam bem para dois casos:

- Maioria dos conjuntos de dados do mesmo domínio possuem *metatargets* similares (AMZ e JT).
- Conjuntos de domínios diferentes (BC e FL), mas como *metatargets* similares.

Problemas:

- Cojuntos do mesmo domínio com *metatargets* diferentes (YH).
- Anomalias: alguns conjuntos de dados estão projetados próximos no espaço sem nenhuma explicação válida para isso (TA)

Trabalhos futuros

- Refazer todos os experimentos com os novos *datasets* de *CF* que foram coletados.
  - *grid search* mais detalhado para as novas técnicas de *embeddings* para grafos utilizadas.
- Treinar os *datasets* usando os novos algoritmos do *framework Elliot* (`https://elliot.readthedocs.io/en/latest/`).
  - *Latent Factor Models*, *Artificial Neural Networks* e *Unpersonalized Recommenders*.
  - Otimização de hiperparâmetros: *Hyperopt*.

References

Brazdil, P. et al (2008)
Metalearning: Applications to data mining.
Springer Science & BusinessMedia.

Cunha, T., Soares, C., and de Carvalho, A. C. (2018)
cf2vec: Collaborative filtering algorithm selection using graph distributed representations
arXiv preprint arXiv:1809.06120

Grover, A. and Leskovec,J. (2016)
node2vec: Scalable feature learning for networks.
Proceedings of the 22nd ACM SIGKDD, 855–864.

Ivanov, S. and Burnaev, E.(2018)
Anonymous walk embeddings.
Proceedings of the 35th International Conference on Machine Learning, 2191–2200.

Ribeiro, L. F., et al (2017)
Struc2vec: Learning node representations from structural identity.
Proceedings of the 23rd ACM SIGKDD, 385–394.

The End