Mini Project Report: Coreset Selection for Efficient Recommendation

Ziye Guo

August 29, 2025

1 Introduction

Recommender systems generally trained on a large dateset. We made use of the MovieLens data set, which contains one million ratings given by people to movies. Training models based on such large data sets cost much computer resources and time, and would cost companies more money on servers or cloud service if they want to store and analyze large volumes of data which affects their budget, as such, in something like movie recommendation where user data is constantly growing, it's not so easy to train a model to meet their need without additional expenses.

Coreset selection picks out a small portion of the data that is still able to offer some benefit to training. This may help shorten the training time needed and save money, but also produce effective recommendations. In this project we check how coreset-based methods work for recommendation models, starting from collaborative filtering: we use simple matrix factorization as well as neural model such as NCF. Our goal here is to find whether we can get similar performance as the whole data when training on these small sets, and discover the difference between various methods'performance.

The idea is based on a paper in RecSys 2024 [Hurley et al., 2024], but we tested more ways and delved into why these methods functions as they do.

2 Problem Formulation

In collaborative filtering (CF) [Su and Khoshgoftaar, 2009], we have a sparse matrix $R \in R^{m \times n}$, where m is the number of users, n is the number of items (e.g., movies), and each entry r_{ij} shows the interaction between user i and item j. This can be an explicit rating (e.g., 1 to 5 stars) or implicit feedback (e.g., click, purchase, or view, often binarized as 1 for interaction and 0 otherwise). We only observe some entries in the set $\Omega \subseteq [m] \times [n]$.

The goal is to estimate the missing entries of R by learning a predictive model from the observed interactions.

Among various CF methods, matrix factorization (MF) [Koren et al., 2009] is a classic baseline. It approximates R by the product of low-rank user and item embeddings $U \in R^{m \times d}$ and $V \in R^{n \times d}$:

$$\hat{r}_{ij} = u_i^T v_j,$$

where u_i and v_j are the i-th and j-th rows of U and V (representing the latent factors for user i and item j), and d is the embedding dimension. MF offers an interpretable linear interaction mechanism that provides a solid foundation for later neural models.

In our implementation, we use a SimpleMF model, which applies a sigmoid activation to the inner product for implicit feedback, but not for explicit feedback:

$$\hat{r}_{ij} = \begin{cases} \sigma\left(u_i^T v_j\right) & \text{(implicit)}, \\ u_i^T v_j & \text{(explicit)}, \end{cases}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function.

We also implement Neural Collaborative Filtering (NCF), which combines Generalized Matrix Factorization (GMF) and Multi-Layer Perceptron (MLP). The GMF part computes:

$$g = u_i^{\text{GMF}} \odot v_j^{\text{GMF}},$$

where $\odot$ denotes element-wise product, and u_i^{GMF}, v_j^{GMF} are separate embeddings for GMF.

The MLP part processes the concatenated embeddings:

$$m = \text{MLP}\left([u_i^{\text{MLP}}, v_j^{\text{MLP}}]\right),$$

where $[\cdot, \cdot]$ is concatenation, and u_i^{MLP}, v_j^{MLP} are embeddings for MLP.

The final prediction is:

$$\hat{r}_{ij} = \begin{cases} \sigma\left(W^T[g, m]\right) & \text{(implicit)}, \\ W^T[g, m] & \text{(explicit)}, \end{cases}$$

where W is the weight vector for the prediction layer.

For training, we distinguish between explicit and implicit feedback. For explicit feedback (ratings), we use mean squared error (MSE) loss:

$$\mathcal{L}_{\text{MSE}}(\theta) = \sum_{(i,j)\in\Omega} (r_{ij} - \hat{r}_{ij})^2 + \lambda(\|U\|_F^2 + \|V\|_F^2),$$

where θ are the model parameters, $\|\cdot\|_F^2$ is the Frobenius norm for regularization, and λ is the regularization coefficient. However, in our code, we simplified by omitting the regularization term and using PyTorch's `nn.MSELoss()` with mean reduction.

For implicit feedback, we set observed $r_{ij} = 1$ and use pointwise binary cross-entropy (BCE) with negative sampling:

$$\mathcal{L}_{\text{BCE}}(\theta) = -\sum_{(i,j)\in\Omega^+} \log \hat{y}_{ij} - \sum_{(i,k)\in\Omega^-} \log(1 - \hat{y}_{ik}) + \lambda\|\theta\|^2,$$

where Ω^+ is the set of positive (observed) pairs, Ω^- is the set of negative samples (4 per positive, randomly sampled), $\hat{y}_{ij}$ is the model's sigmoid-predicted probability, and $\lambda\|\theta\|^2$ is L2 regularization (omitted in our code for simplicity; we use `nn.BCELoss()` with mean reduction). Note that while pairwise ranking losses like Bayesian Personalized Ranking (BPR) could be used:

$$\mathcal{L}_{\text{BPR}}(\theta) = -\sum_{(i,j,k)} \log \sigma(\hat{y}_{ij} - \hat{y}_{ik}) + \lambda\|\theta\|^2,$$

where (i, j, k) are user-positive-negative triplets, our implementation sticks to pointwise BCE.

Coreset selection aims to find a subset $S \subset \Omega$ such that training on S approximates the performance of training on the full Ω, reducing computational costs.

3 Methods

In order to deal with training problems posed by huge amounts of data, we try to select coresets, which is aiming at choosing a representative small subset $S \subset \Omega$ from the dataset in order to be able to train the models while having much smaller size and good generalization, without reducing much the training set. The techniques used forr constructing coresets for recommender systems [Hurley et al., 2024] includes an approach using both clustering and gradient techniques. We extend this by combining with other methods like the greedy k-center selection methods, and introduce variants that do not use pretraining to evaluate whether interaction value is something that can be heuristically estimated versus requiring model driven signals (like pretraining). Furthermore, these choices give us access to ways of quantifying representativeness that are more similar to those used in collaborative filtering and ones we dive deeper into in our analysis section (i. e. ways of quantifying based on gradient norms, embedding diversity etc.).

All methods select a coreset of size approximately $r \times |\Omega|$, where $r = 0.2$ is the default ratio. For advanced methods (clustering, gradient, greedy), we use a pretrained SimpleMF model to derive user embeddings or gradients, unless specified as no-pretrain (random initialization). This pretraining is simplified to 5 epochs without regularization, focusing on efficiency. The coresets are then used to train the main model (SimpleMF or NCF).

Algorithm 1 Forward Pass of SimpleMF Model

User indices u, Item indices i.
$u_{embed} \leftarrow$ user_embed(u)
$i_{embed} \leftarrow$ item_embed(i)
$out \leftarrow \sum(u_{embed} \odot i_{embed})$
if feedback_type = 'implicit' **then**
  $out \leftarrow \sigma(out)$ % Sigmoid for implicit feedback
else
  $out \leftarrow out$ % Linear output for explicit
end if
return out

3.1 Random Sampling (Baseline)

By far, the simplest approach is random sampling; it simply samples random interactions from Ω. It can be considered a trivial baseline to contrast against other approaches and has been a particularly strong baseline on coreset studies of recommender systems [Hurley et al., 2024]. In practice we directly sample indices without replacement so as not to have any kind of preference toward either user or item.

3.2 Clustering-based Selection

We use clustering-based coresets to cluster similar users or items and take one representative point from each cluster to guarantee diversity. Following Hurley et al. [2024], we apply K-means clustering ($K = 10$) on user embeddings from a pretrained SimpleMF. Then in each cluster we choose the nearest neighbors of the cluster centroids using approximate nearest neighbors (ANN), which promote the representativity of selected points.

The no-pretrain variant clusters random-initialized embeddings; it allows us to test the power of heuristics (pretrain-free) vs. model driven signals. By focusing only on users, this simplification

omits item embeddings as is typical for user-centric recommender systems [Jiang et al., 2020].

3.3 Gradient-based Selection

Gradient-based methods favor the most interactive values, where the sensitivity to the model parameters is determined based on gradient norms. Inspired by Hurley et al. [2024] and general coreset literature [Mirzasoleiman et al., 2020], we pre-train SimpleMF and use per-interaction gradients to determine which top-r fraction has highest norms.

The no-pretrain version uses gradients from a random model, serving as a heuristic baseline to compare with model-driven pretraining. We simplify by focusing on user embedding gradients and subsample if needed for efficiency, differing from full-batch computations in prior work.

Algorithm 2 Gradient Coreset Selection (with or without Pretraining)

Dataset, number of users num_u number of items num_u, ratio and Losstype.
mf ← SimpleMF(num_u, num_i)
if pretrain **then**
mf_loader ← DataLoader(*dataset*, batch_size=256, shuffle=True)
mf, _ ← train_model(mf, mf_loader, num_i, epochs=5)
end if
user_embeds ← mf.user_embed.weight.detach().cpu().numpy()
gradients ← []
(u, i, l): user index, item index, label (rating or 0/1) from dataset
for (u, i, l) in DataLoader(*dataset*, batch_size=1) % per sample. **do**
    pred ← mf(u, i)
    if *Losstype* = 'MSE' **then**
loss ← MSELoss(pred, l) % MSE loss for explicit
    else
loss ← BCELoss(pred, l) % BCE loss for implicit
    end if
    loss.backward()
    grad_norm ← norm(mf.user_embed.weight.grad[u.item()])
    gradients.append(grad_norm)
    mf.zero_grad()
end for
indices ← argsort(gradients)[-int(len(*dataset*) * *ratio*):]
return Subset(*dataset*, indices)

3.4 Greedy k-center Selection

Greedy k-center selection increases diversity as each iteration chooses the point that is furthest from the previously selected one in order to minimize the largest distance to any point [Sener and Savarese, 2018]. We adopt this approach to recommender systems and use it on top of the pre-trained user embeddings to pick a subset of users and their interactions to form the coreset. Diversity is then measured.

In the no-pretrain variant, using random embeddings makes it possible to adopt a heuristic approach. In our implementation, we use subsampling to avoid constructing pairwise distance matrices on huge data sets, and make trunction of the selection result such that it is of exact size for simplicity of applying the original guarantee on practical recommender systems.

4 Experiments

To test the usefulness of the coreset selection methods, we use the MovieLens 1M data set as an experiment sample for both explicit and implicit feedback environments. In addition to measuring the performance of recommending items and system computation speed, these experiments provide more accurate understanding of how the coreset selection methods work.

4.1 Experimental Setup

We used the MovieLens 1M dataset which contains about 1 million ratings from 6,040 users on 3,952 movies for our experiments. The data preprocessing follows common practices where users and items with fewer than 5 interactions are filtered out and binary ratings are used for the implicit feedback. We use a leave-one-out method for splitting the data. Namely, for each user, the last interaction by time-stamp is used as test while one randomly sampled interaction from the remaining is set aside for validation, the rest used for training.

To sum up, we select embedding size 32 for SimpleMF as the single-model for explicit ratings training and use NCF for all models. We use Adam to train our models with a learning rate of 0.001, batch size of 256, and 20 epochs. When doing explicit ratings training we employ MSE loss, whereas for implicit feedback we have used BCE with four negative samples for each positive sample. We take RMSE as the evaluation metric for explicit ratings (the lower, the better) and recall@10 and NDCG@10 for implicit ratings (the higher, the better, 99 negative samples are chosen for each positive sample). Coreset ratio is set to 0.2 and simple MF pretraining using simple MF (five epochs) is applied when necessary. All experiments were run on a regular CPU/GPU setup and given in seconds. For insight analyses (Q1-Q5), visualizations and their related metrics (gradient norm, coverage, and correlation) are explored.

4.2 Results

Table 1 reports results for SimpleMF on explicit feedback. For NCF, Tables 2 and 3 show the results of NCF on explicit feedback and on implicit feedback respectively. Figures 1–5 visualize the selected insights.

Table 1: Results for SimpleMF on Explicit Feedback (MovieLens 1M).

Method	RMSE	Select Time	Train Time	Total Time
Full	2.685	0	414.96	414.96
Random	2.815	0.02	85.76	85.78
Gradient_no_pretrain	2.806	644.70	84.86	729.56
Greedy_no_pretrain	2.786	2.03	94.51	96.54

4.3 Analysis

For explicit feedback with SimpleMF (Table 1), random sampling achieves a reasonable RMSE (2.815) with minimum pick times. While Gradient no pretrain and greedy no pretrain achieve comparably acceptable RMSE(2.806 and 2.786) is either consumes more time or performs worse. No-pretrain variant was solely applied in this case.

With NCF on explicit feedback (Table 2), full training yields the best RMSE (0.984), but coresets like random (1.060) and gradient_no_pretrain (1.064) approach it with 5x time savings.

Table 2: Results for NCF on Explicit Feedback (MovieLens 1M).

Method	RMSE	Select Time	Train Time	Total Time
Full	0.984	0	723.42	723.42
Random	1.060	0.02	146.86	146.88
Clustering	1.694	114.41	1.17	115.58
Gradient	1.332	794.79	147.46	942.25
Gradient_no_pretrain	1.064	684.87	146.18	831.05
Greedy	1.101	115.08	162.71	277.79
Greedy_no_pretrain	1.110	2.20	163.22	165.42

Table 3: Results for NCF on Implicit Feedback (MovieLens 1M).

Method	Recall@10	NDCG@10	Select Time	Train Time	Total Time
Full	0.582	0.320	0	1049.03	1049.03
Random	0.441	0.237	0.02	215.61	215.63
Clustering	0.165	0.079	139.05	1.68	140.73
Gradient	0.104	0.047	878.58	213.84	1092.42
Gradient_no_pretrain	0.330	0.183	738.03	212.88	950.91
Greedy	0.449	0.242	139.59	235.81	375.40
Greedy_no_pretrain	0.449	0.245	2.10	235.88	237.98

Pretrained-based models such as gradient (1.332) do not perform well and might be affected by overfitting when selecting good choices; while the pretraining-free ones have the best efficiency–performance tradeoff.

For implicit feedback using NCF in Table 3, both greedy and greedy-no-pretrain attain close-to-random results (Recall@10 0.449), but enjoy substantial time savings (375 seconds and 238 seconds relative to 1,049 seconds full). Without pretrained model (0.330)gradient outperforms (0.104) gradient; this means that heuristic based signal may outperform model-driven signal when data is sparse.

Visualizations also help show the design of our methods. Figure 1 shows that the gradient no-pretrain coreset is biased towards active users. Embedded norm histogram of Figures 2 and 3 show very low Pearson or Kendall correlation between pre-train and non-pretrain scores. The scatter plots from Fig. 4 to Fig. 5 show diversity, where pre-train embeddings can cluster and leave few redundancies.

Generally, no-pretrain variants are more efficient with less impact on model performance compared to those pre-trained ones, and our experiments indicate that the heuristics are superior to models in exploring trade-offs, so future works may tune the ratios or scale up the dataset.

5 Conclusion

In this work we explored coreset selection techniques to train recommender systems more efficiently while using collaborative filtering (NCF, SimpleMF), and found that using coresets could decrease training time up to 5x, while still maintaining reasonable performance especially without pre-training for implicit feedback datasets.

Limitations include long gradient method selection time, only one data set evaluated (MovieLens 1M) and no comprehensive hyperparameters tuning. Other works might apply different datasets,

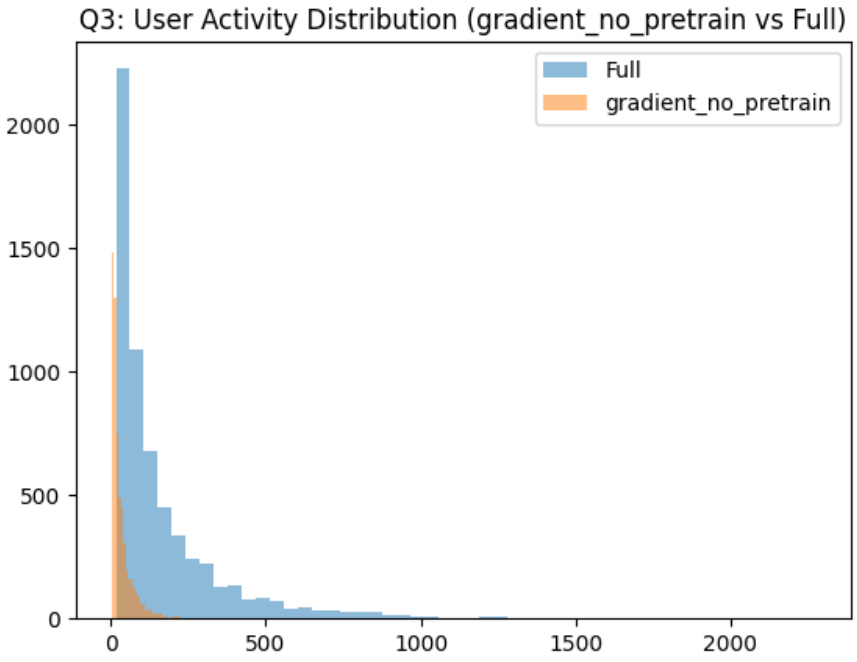

Figure 1: Q3: User Activity Distribution (gradient_no_pretrain vs Full). The coreset favors highly active users compared to the full dataset.

investigate more coresets methods (e.g., dynamic during training), and define adaptive ratio for better result in terms of efficiency.

References

Neil J. Hurley et al. Exploring coresets for efficient training and consistent evaluation of recommender systems. In *Proceedings of the 18th ACM Conference on Recommender Systems*, 2024. URL `https://dl.acm.org/doi/10.1145/3640457.3691716`.

Jyun-Yu Jiang, Patrick H. Chen, Cho-Jui Hsieh, and Wei Wang. Clustering and constructing user coresets to accelerate large-scale top-k recommender systems. In *Proceedings of The Web Conference 2020*, pages 2177–2187, 2020. doi: 10.1145/3366423.3380283.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. URL `https://proceedings.mlr.press/v119/mirzasoleiman20a.html`.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A coreset approach. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=H1aIuk-RW`.

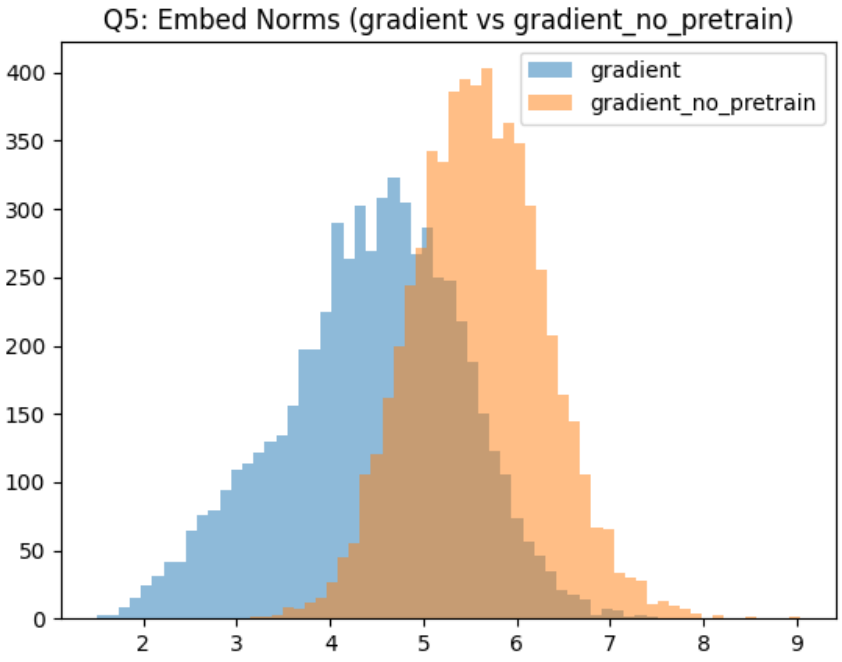

Figure 2: Q5: Embed Norms (gradient vs gradient_no_pretrain). Low correlation suggests heuristic signals differ from model-driven ones.

Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009, 2009.

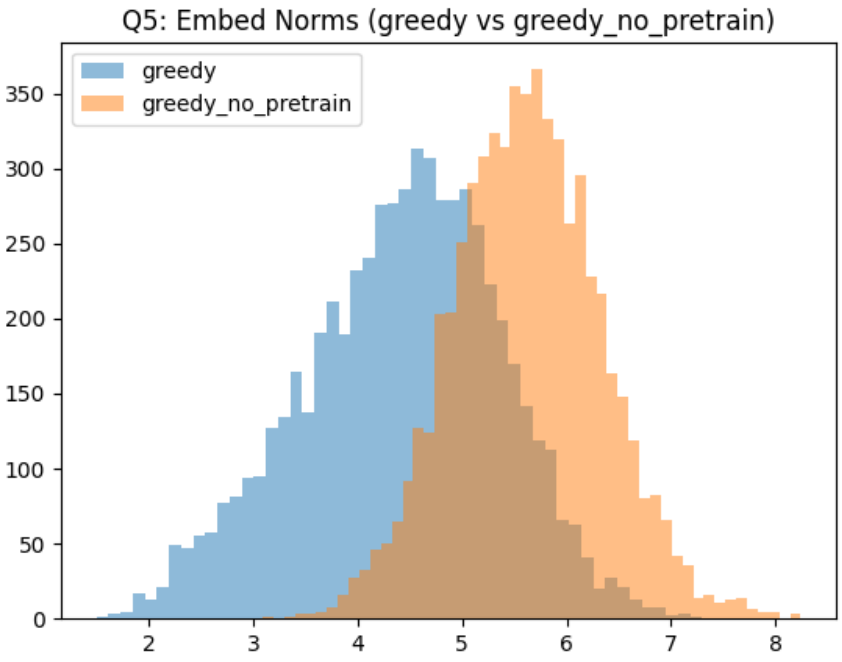

Figure 3: Q5: Embed Norms (greedy vs greedy_no_pretrain). Similar to gradient, indicating pretrain impacts value estimation.

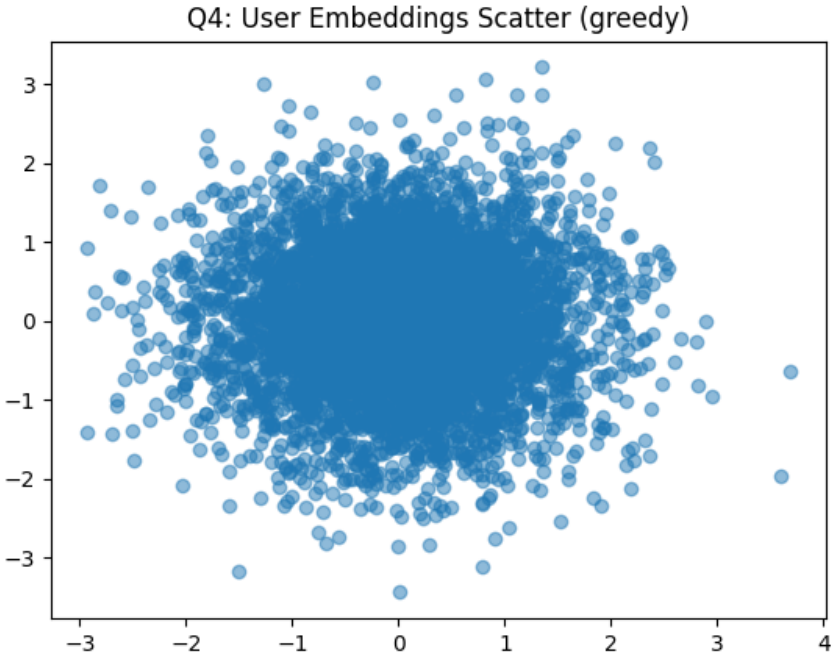

Figure 4: Q4: User Embeddings Scatter (greedy). Shows clustered diversity in the coreset embeddings.

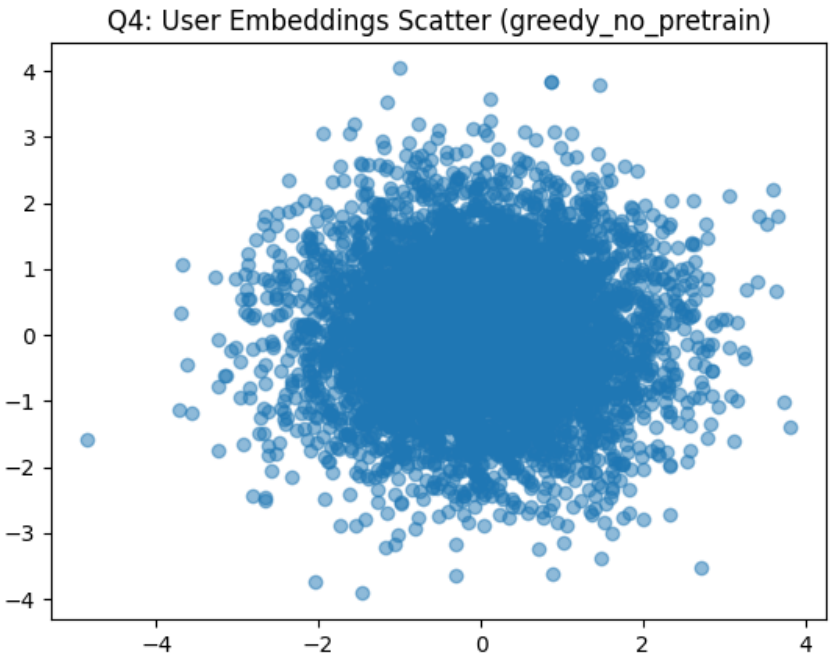

Figure 5: Q4: User Embeddings Scatter (greedy_no_pretrain). Less structured than pretrained, reflecting heuristic limitations.