Fast and Accurate User Cold-Start Learning Using Monte Carlo Tree Search

Dilina Chandika Rajapakse*
Trinity College Dublin
Dublin 2, Ireland
rajapakd@tcd.ie

Douglas Leith*
Trinity College Dublin
Dublin 2, Ireland
doug.leith@tcd.ie

ABSTRACT

We revisit the cold-start task for new users of a recommender system whereby a new user is asked to rate a few items with the aim of discovering the user's preferences. This is a combinatorial stochastic learning task, and so difficult in general. In this paper we propose using Monte Carlo Tree Search (MCTS) to dynamically select the sequence of items presented to a new user. We find that this new MCTS-based cold-start approach is able to consistently quickly identify the preferences of a user with significantly higher accuracy than with either a decision-tree or a state of the art bandit-based approach without incurring higher regret i.e the learning performance is fundamentally superior to that of the state of the art. This boost in recommender accuracy is achieved in a computationally lightweight fashion.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

User cold start, Monte Carlo Tree Search

ACM Reference Format:
Dilina Chandika Rajapakse and Douglas Leith. 2022. Fast and Accurate User Cold-Start Learning Using Monte Carlo Tree Search. In *Sixteenth ACM Conference on Recommender Systems (RecSys '22), September 18–23, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3523227.3546786

1 INTRODUCTION

In this paper we revisit the cold-start task for new users of a recommender system, which remains a core challenge. When a new user joins the system it initially has no knowledge of the preferences of the user and so would like to quickly learn these[1]. The recommender system therefore initially starts in an "exploration" phase where the first few items that it asks the new user to rate are chosen with the aim of discovering the user's preferences. We focus on the simplest setup where a user explicitly rates items presented to them, e.g. on a 1-5 scale or binary like/dislike feedback, and the aim of the recommender system is to predict other items that the user may like.

One common approach to this new user cold-start task is to take ratings already collected from a population of users, use these to cluster users into groups and then train a decision-tree to learn a mapping from item ratings to the user group, see for example Figure 1(a). When a new user joins the system this decision-tree is used to decide which items the user is initially asked to rate and in this way the group to which the user belongs is initially estimated. Once the group is estimated, the system recommends items liked by members of that group e.g. using matrix factorisation or another collaborative filtering approach.

However, typically users clustered in the same group do not give identical ratings to an item. Rather there is a spread of ratings, and this intra-cluster variability between users can be thought of as adding noise to the ratings. Unfortunately, decision trees can easily make mistakes in the face of such noise. For example, Figure 1(b) shows the measured decision-tree accuracy for Netflix data clustered into 16 groups (see later for more details). It can be seen that the accuracy is as low as 50-60% for a number of groups.

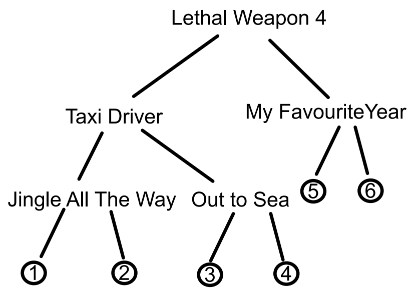
The user cold-start task can be viewed as a form of single-player game. In a single-player game a sequence of moves is selected with the aim of maximising the reward or score generated by an environment (which may have some randomness). Supposing t moves have already been made, then lookahead to the next sequence of d moves can be represented by a path in a tree of depth d. Selecting the next move then involves exploring this tree to find a good future sequence of moves and then making the first move from that sequence, and the process then repeats starting from move $t + 1$.

We can directly map this to user cold-start as follows. A move consists of presenting a user with an item and observing their rating. After t moves we have presented a user with t items and observed t ratings. The next sequence of d moves consists of a path in a tree of depth d where each node is an item. The score of a sequence is the estimated probability of learning the correct user group by presenting that sequence of items to the user (we discuss the details of this calculation below). The first item of the sequence of d items with the highest score is then presented to the user, and the process then repeats. Since we have a budget of t_{max} items, the maximum lookahead d at step t is equal to $t_{max} - t$.

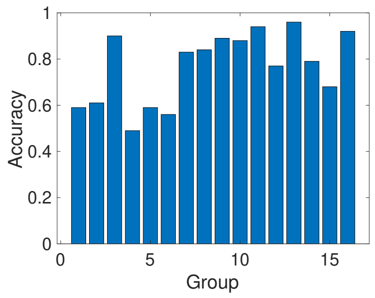
Observe that selecting the next move/item-to-present involves searching a lookahead tree of depth j. Monte Carlo Tree Search (MCTS) replaces exhaustive exploration of this tree with targetted exploration of only the most promising paths. This allows deeper tree exploration and better solutions to be found, even when the

*Both authors contributed equally to this research.
[1]The system may have some general context regarding the user, such as the country/city they are located in, user gender and demographics *etc*, in which case learning is conditioned on this but the fundamental cold start task otherwise remains unchanged.

RecSys '22, September 18–23, 2022, Seattle, WA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9278-5/22/09.
https://doi.org/10.1145/3523227.3546786



(a)



(b)

Figure 1: (a) Illustrating a movie recommender decision-tree (adapted from [18]), (b) Decision-tree accuracy for Netflix data (16 groups).

tree branching factor (the number of possible next moves/items) is large. MCTS has now largely replaced A/B trees in computer game playing and has contributed to breakthrough performance in Go (e.g. see AlphaGo [16]) and Chess (e.g. see [17]).

We find that an MCTS-based approach to cold-start is able to achieve extemely fast learning of user preferences. We demonstrate that the group of a user is consistently quickly identified with significantly higher accuracy than with either a decision-tree or a state of the art bandit-based approach without incurring higher regret i.e the learning performance is fundamentally superior to that of the state of the art. This boost in recommender accuracy is achieved in a computationally lightweight fashion, with our MCTS-based implementation able to perform over 30+ recommendations per second using a single CPU core for the Netflix dataset with 8 distict user groups. It is trivially parallelisable and so linearly scalable with the number of CPU cores.

2 RELATED WORK

For a recent survey of solutions to the cold-start see [7, 8]. Passive approaches include recommending popular items, use of item-based recommendation (once user starts rating items an item-based approach is used to recommend similar items), transfer learning from another recommender system previously used by a user, and asking new users to rate a fixed list of items. Examples of early work on active learning include IGCN (information gain through clustered neighbors) which uses a decision tree with user clusters as leaves [11] and the ternary decision-tree approach of [10]. More recently, [1] uses representative items i.e. after completing the ratings matrix R, k columns of R are selected, the ratings of the other items are represented as a linear combination of these and during cold start a new user is asked to rate these representative items. This approach is extended to use a decision-tree approach by [15]. In [18] a matrix factorization approach is proposed whereby a decision-tree is trained to map from item ratings to the latent feature vector for a user.

Use of multi-arm bandits (MABs) for user cold-start has also received attention. In [9] after completing the ratings matrix R its rows are clustered and the average ratings vector for each cluster is used as a representative user. During cold start an MAB is used to select the average ratings vector to use and the user is asked to rate the next highest item in the vector. In [3] a MAB is used to select between recommender strategies, typically recommending popular items initially for a new user and later switching to a kNN or matrix factorisation model. Note that naive application of standard bandit algorithms to the cold-start task leads to poor performance. If we think of each recommender system item as an arm of a MAB then we run into the difficulty that (i) there are many arms and so learning is slow and (ii) repeated pulls of the same arm tend to be highly correlated. One remedy is to associate an arm with each group rather than each item [9]. For each group the available items are sorted in descending order of their predicted rating by users in that group. Pulling the arm for a group then corresponds to asking the user to rate the next item from this sorted list, i.e. the unrated item predicted to have the highest rating for members of the group. While this greatly reduces the number of arms in the MAB, the learning rate remains very slow [14]. This is because items rated highly by members of one group tend to also be rated highly by members of at least some of the other groups, and so the user ratings for these items do not serve to strongly distinguish between groups and so allow rapid learning. To address this, [14] identify so-called distinguisher items that tend to have distinct ratings by users in different pairs of groups. Using these they propose a Cluster-based Bandit algorithm that we use as a baseline for comparison in the present paper.

Contextual bandits have also received attention for user cold-start. These make use of contextual information, e.g. the user's location, gender, demographics. In this paper we assume that such contextual information is absent but using similar techniques to contextual bandits our approach can be readily extended to take advantage of it when it is available, see Section 5.

Monte Carlo Tree Search was introduced by [5] and quickly adopted within the two-player game community. See [2] for a survey of MCTS methods developed in the first 5 years after its introduction. MCTS was used by AlphaGo [16] to defeat the world champion in the game of Go, and also in the later AlphaZero [17] game playing engine. Use of MCTS in single player games was introduced in [12, 13]. For more recent work, see for example [6] and references therein.

3 MONTE CARLO TREE SEARCH FOR USER COLD-START

3.1 Preliminaries

We have a set $\mathcal{G}$ of user groups. Each user belongs to one group $g \in \mathcal{G}$. We also have a set of items $\mathcal{V}$. Given a new user our task is to quickly learn which group they belong to by asking the user to rate t_{max} items in $\mathcal{V}$, using the fact that the distribution of item ratings varies depending on the user's group.

Let $\mathcal{V}^{(t)} = \{v_1, \ldots, v_t\}$ with $v_i \in \mathcal{V}$, $i = 1, \ldots, t$ be the set of items rated by a user and $R(v_i)$ be the user's rating of item v_i. Initially, for a new user $t = 0$ and $\mathcal{V}^{(0)}$ is the empty set. Let $p_g^{(t)}, g \in \mathcal{G}$ be an estimate of the probability that the user belongs to group g given the the user has rated the items $\mathcal{V}^{(t)}$, with $\sum_{g \in \mathcal{G}} p_g^{(t)} = 1$ and $0 \le p_g^{(t)} \le 1$. When $t = 0$ these probabilities can be initialised to the uniform distribution $p_g^{(0)} = 1/|\mathcal{G}|$, or alternatively to a distribution derived from population data.

Our MCTS approach is agnostic to how the probabilities $p_g^{(t)}$ are calculated, but for concreteness in our examples we will assume that for users belonging to group g the rating $R(v)$ of item v is i.i.d. gaussian with mean $\mu(g, v)$ and variance $\sigma^2(g, v)$. Denoting the user's group by random variable G we then have that

$$p(R(v) = r|G = g) = (1/\sqrt{2\pi}\sigma(g, v))e^{-(r-\mu(g,v))^2/2\sigma^2(g,v)}$$

and for observed sequence $D^{(t)}$ of ratings $R(v_1), R(v_2), \ldots, r(v_t)$ for items $v_1, v_2, \ldots, v_t$ it follows that

$$p(D^{(t)}|G = g) = \gamma^{(t)}(g)e^{-L^{(t)}(g)}$$

where $L^{(t)}(g) := \sum_{i=1}^{t}(R(v_i) - \mu(g, v_i))^2/2\sigma^2(g, v_i)$, $\gamma^{(t)}(g)) := 1/(2\pi)^{t/2} \times 1/\Pi_{i=1}^{t}\sigma(g, v_i)$. By Bayes rule,

$$p_g^{(t)} = p(G = g|D^{(t)}) = \frac{p(D^{(t)}|G = g)p(G = g)}{p(D^{(t)})}$$
$$= \frac{p(D^{(t)}|G = g)p(G = g)}{\sum_{h \in \mathcal{G}} p(D^{(t)}|G = h)p(G = h)}$$

with $p(D^{(t)}) = \sum_{h \in \mathcal{G}} p(D^{(t)}|G = h)p(G = h)$. Assuming uniform prior $p_g^{(0)} = p(G = g) = 1/|\mathcal{G}|$ then

$$p_g^{(t)} = \frac{\gamma^{(t)}(g)e^{-L^{(t)}(g)}}{\sum_{h \in \mathcal{G}} \gamma^{(t)}(h)e^{-L^{(t)}(h)}} \tag{1}$$

for $t = 1, 2, \ldots$.

3.2 Efficiently Searching The Lookahead Tree

Suppose at step t a new user has rated items $\mathcal{V}^{(t)}$. The lookahead tree at step t embodies all item sequences of length $d = t_{max} - t$ where t_{max} is the total number of cold start items to be presented to a new user, e.g see Figure 2. Note that we only ask a user to rate a given item once since repeated ratings of the same item tend to be highly correlated[2]. To select the next item to present to the user our MCTS-based approach proceeds by repeatedly executing the following actions: (i) select a sample lookahead path $\mathcal{S} = \{s_{t+1}, \ldots, s_{t_{max}}\}$ of $d = t_{max} - t$ items from the tree, (ii) draw a random user group $\hat{g}$ according to probability distribution $\{p_g^{(t)}, g \in \mathcal{G}\}$, (iii) draw synthetic ratings $R(s_{t+1}), \ldots, R(s_{t_{max}})$ for a user from group $\hat{g}$, (iv) use $\mathcal{V}^{(t)}$ and synthetic ratings $R(s_{t+1}), \ldots, R(s_{t_{max}})$ to estimate the group probabilities $p_g^{(t_{max})}$, (v) if $\hat{g}$ is the group with highest probability generate reward +1 (the user group is correctly identified) else generate reward 0, (vi) backpropagate this reward along path $\mathcal{S}$ in the lookahead tree. We give more details on these steps below.

In principle, in step (i) we would like to select a different path at every turn until all paths in the lookahead tree have been visited. We can then select the path with highest reward, present the first item in that path to the user, observe the user's rating, update t to $t + 1$, $\mathcal{V}^{(t)}$ to $\mathcal{V}^{(t+1)}$ and repeat the MCTS steps. However, the lookahead tree is generally far too large for an exhaustive search over all paths to be feasible. To see this observe that in a lookahead tree of depth d at time $t = 0$ there is a root node with $|\mathcal{V}|$ children (corresponding to each possible item in set $\mathcal{V}$), and each child in turn has $|\mathcal{V}| - 1$ children and so on. The tree therefore has $|\mathcal{V}|!/d!(|\mathcal{V}| - d)!$ leaf nodes and when, for example, $|\mathcal{V}| = 1000$ and $d = 5$ then the tree has around 10^{12} leaf nodes. Rather than trying to carry out an exhaustive search of the lookahead tree, instead in step (iii) we try to identify good paths that are likely to generate high reward and focus on exploring these.

3.2.1 *Finding Good Paths.* Since the full lookahead tree may be extremely large our aim is to construct a subtree containing paths that tend to generate high reward. Each node n in this subtree lies at the end of a path (i.e. sequence of items) $\mathcal{S}(n)$ that leads from the tree root to the node. Associated with each node n in the tree is: (i) an item $s(n)$ (namely, the last item in path $\mathcal{S}(n)$), (ii) a count $N(n)$ of the number of times that $s(n)$ has been included in the MCTS lookahead set $\mathcal{S}$ and (iii) a reward $Q(n)$ which is the sum of the rewards generated by lookahead sets $\mathcal{S}$ that include $s(n)$.

We construct this subtree (and in the process also generate sample lookahead paths) as follows. Initially create a tree consisting of a root node and $|\mathcal{V}|$ child nodes i.e. each child node is associated with a different item in $\mathcal{V}$. Initialise counters $N(\cdot) = 0$ and $Q(\cdot) = 0$ for these child nodes. Pick a child node n uniformly at random and set item $s_{t+1} = s(n)$. We need to extend this to obtain a sequence $\mathcal{S}$ of d items, and the simplest way to do this is just to pick $d - 1$ items $s_{t+2}, \ldots, s_{t_{max}}$ uniformly at random from $\mathcal{V}$. Generate the reward for this path $\mathcal{S}$ and add this to counter $Q(n)$ associated with the selected child node, also increment the $N(n)$ counter associated with the child node. At the next turn, select a child node uniformly at random from the child nodes with $N(\cdot) = 0$ and repeat this process.

Once there are no child nodes with $N(\cdot) = 0$, pick a child node n with high reward, set item $s_{t+1} = s(n))$ and add $|\mathcal{V} \setminus \{s(n)\}|$ child nodes to this node. Select one of these children uniformly at random set this as item s_{t+2}. Extend sequence $\{s_{t+1}, s_{t+2}\}$ to obtain a sequence $\mathcal{S}$ of length d by selecting $d - 2$ items uniformly at random. Generate the reward for this path $\mathcal{S}$ and add this to the counters $Q(\cdot)$ of the nodes corresponding to items s_{t+1} and s_{t+2}, also increment the $N(\cdot)$ counters associated with these nodes.

At the next turn, repeat this process. That is, traverse the current tree picking nodes with high reward until either a leaf node is

[2]Measurement studies indicate that when people are repeatedly asked to rate the same item on a scale of 1-5 then if they rate 1 or 5 they tend to consistently stick with that rating although when they rate 3 or 4 they may change their rating back and forth between 3 and 4. That is, lumping ratings of 3-4 together in a single bucket these previous studies indicate that a user's rating of an item tends to be consistent.

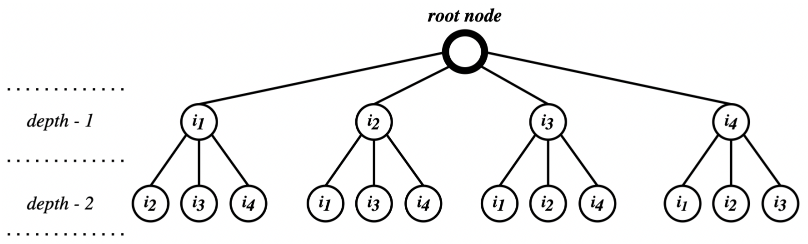

root node
depth - 1
depth - 2
i_1 i_2 i_3 i_4
i_2 i_3 i_4 i_1 i_3 i_4 i_1 i_2 i_4 i_1 i_2 i_3

Figure 2: Example of lookahead tree with $|\mathcal{V}| = 4$, $d = 2$

reached or a node with unvisited children. If a leaf node n is reached then expand it by adding $|\mathcal{V} \setminus \mathcal{S}(n)|$ child nodes, and pick one of these at random. If a node with unvisited children is reached, pick uniformly at random from the unvisited childen. This yields a partial sequence of items, of length equal to the depth of the tree. This is then extended to a sequence $\mathcal{S}$ of length d by selecting the remaining items uniformly at random. Generate the reward for this path $\mathcal{S}$ and update the $Q(\cdot)$ and $N(\cdot)$ counters of the nodes in the tree on path $\mathcal{S}$.

Algorithm 1 gives pseudo-code for this process of growing a lookahead subtree containing paths that tend to generate high reward.

Algorithm 1 MCTS For User Cold-Start

```
$\mathcal{V}^{(t)} \leftarrow \emptyset$
for $t = 0; t = t + 1; t \leq t_{max}$ do
    Initialise lookahead tree $\mathcal{T} \leftarrow$ root node
    for turn $i = 1; i = i + 1; i \leq max\_turns$ do    ▷ Construct lookahead subtree
        $j = t + 1$
        $n_j \leftarrow best\_child(\mathcal{T}_{root}); s_j = s(n_j)$    ▷ Select good path
        while ($n_j$ != null) and ($j < t_{max}$) do
            $n_{j+1} \leftarrow best\_child(n_j); s_{j+1} = s(n_{j+1})$
            $j \leftarrow j + 1$
        end while
        if ($n_j$ is leaf node) and ($j < t_{max}$) then    ▷ Expand
            Add $|\mathcal{V} \setminus \mathcal{S}(n_j)|$ children to $n_j$, initialise children's $Q(\cdot)$, $N(\cdot)$ to 0
            $n_{j+1} \leftarrow best\_child(n_j); s_{j+1} = s(n_{j+1})$
            $j \leftarrow j + 1$
        end if
        while $j < t_{max}$ do    ▷ Randomised rollout
            $s_{j+1} \leftarrow$ select random item
            $j \leftarrow j + 1$
        end while
        $Q \leftarrow calculate\_reward(\mathcal{V}^{(t)}, \{s_{t+1}, \dots, s_{t_{max}}\})$    ▷ Generate reward
        for node $n$ in path $\mathcal{S}$ do    ▷ Backpropagate reward
            $N(n) \leftarrow N(n) + 1; Q(n) \leftarrow Q(n) + Q$
        end for
    end for
    $n^* \leftarrow \arg\max_{n \in children(\mathcal{T}_{root})} N(n)$ (break ties randomly)    ▷ Pick most visited child of tree root
    $v_{t+1} \leftarrow s(n^*)$    ▷ Ask user to rate corresponding item
    $\mathcal{V}^{(t+1)} \leftarrow \mathcal{V}^{(t)} \cup \{v_{t+1}\}$
end for
```

3.2.2 Selecting a Node With High Reward. Once the tree has started to grow, the process described above requires selecting a sequence of child nodes with high reward until a leaf node is reached. In doing this we would like to balance exploration (trying new items) and exploitation (selecting an existing item) and so we adopt an optimistic upper confidence bound (UCB) approach. Namely, we estimate the reward of a node n that is a child of parent node p as

$$Reward(n) = \underbrace{\frac{Q(n)}{N(n)}}_{exploit} + \underbrace{\sqrt{0.25\frac{\log(N(p))}{N(n)}}}_{explore}$$

The first term $Q(n)/N(n)$ is the average reward observed so far for sequences that include item $s(n)$ (i.e. node n). However, when the number of samples $N(n)$ on which this average reward is based is small then the value may well be inaccurate. The second term aims to compensate for this. Intuitively, when $N(n)$ is small the second term is large, encouraging exploration of node n. As $N(n)$ grows, however, the second term becomes smaller and this captures the fact that the first term $Q(n)/N(n)$ is then more reliable. Another way to derive the expression for $Reward(n)$ is to note that the reward is a Bernoulli random variable taking value 0 or 1. Applying Hoeffding's inequality, $P(\frac{Q(n)}{N(n)} \geq \mu + x) \leq e^{-2x^2 N(n)}$. Plugging in $x = \sqrt{0.25\frac{\log(N(p))}{N(n)}}$ then $e^{-2x^2 N(n)} = e^{-0.5\log(N(p))} = 1/\sqrt{N(p)}$. Hence, $Reward(n)$ can be thought of as the upper limit of a confidence interval which increases in power as the number of times $N(p)$ that the parent node is visited grows.

Algorithm 2 gives pseudo-code using this UCB approach to select a child node with high reward.

3.2.3 Calculating The Reward. Given the items $V^{(t)} = \{v_1, \dots, v_t\}$ already rated by a user, plus a sample path $\{s_{t+2}, \dots, s_{t_{max}}\}$ from the lookahead tree, we need to calculate the reward associated with the sequence of items $\{v_1, \dots, v_t, s_{t+2}, \dots, s_{t_{max}}\}$. We have the user ratings for items $v_1, \dots, v_t$ but we lack ratings for items $s_{t+2}, \dots, s_{t_{max}}$.

We therefore adopt a Monte Carlo sampling approach. Namely, we select a user group $\hat{g}$ according to probability distribution $\{p_g^{(t)}, g \in \mathcal{G}\}$ i.e. according to our best estimate of the user group at step t. Initially, we are unsure of the user group and $\{p_g^{(0)}\}$ is the uniform distribution, but as the user rates items we hope that the distribution $\{p_g^{(t)}\}$, and so $\hat{g}$, will start to concentrate on the true group of the user. This concentration behaviour can be seen, for example, in Figure 3 which plots $\{p_g^{(t)}, g \in \mathcal{G}\}$ vs the number of

Algorithm 2 best_child

```
Input: node $p$ of lookahead tree $\mathcal{T}$
$C \leftarrow$children of $p$
if $C == \emptyset$ then
    return null                                    ▷ No child nodes
else
    $C_{unvisited} \leftarrow \{n \in C : N(n) = 0\}$    ▷ Set of unvisited children of node $p$
    if $C_{unvisited} \neq \emptyset$ then
        return child selected uniformly at random from $C_{unvisited}$
    else
        return $\arg\max_{n \in C} \frac{Q(n)}{N(n)} + \sqrt{0.25 \frac{\log(N(p))}{N(n)}}$ (break ties randomly)    ▷ UCB
    end if
end if
```

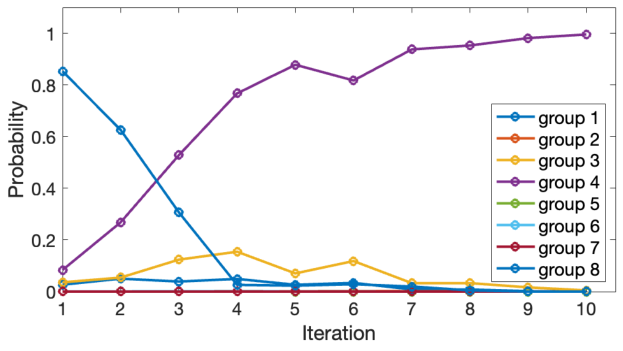


Figure 3: Example illustrating evolution of probabilities $p_g^{(t)}$, $g = 1, 2, \ldots, 8$ vs the number of items t rated by a new user in group $g = 4$. Netflix dataset, $|\mathcal{G}| = 8$ user groups. It can be seen that after rating $t = 3$ items the estimated probability $p_4^{(t)}$ of the user being in group 4 is higher than that of the other groups and $p_4^{(t)} \longrightarrow 1$ as t increases.

items t rated by the user for the Netflix dataset with $|\mathcal{G}| = 8$ user groups. In this example the new user belongs to group $g = 4$ and $p_4^{(t)} \longrightarrow 1$ as t increases while $p_g^{(t)} \longrightarrow 0, g \neq 4$.

Given user group $\hat{g}$, we can generate synthetic ratings $R(s_{t+2}), \ldots, R(s_{t_{max}})$ by making a draw from the multivariate Gaussian distribution with mean $\mu(\hat{g}, v)$ and variance $\sigma^2(\hat{g}, v)$, $v \in \mathcal{V}$ of ratings by users in group $\hat{g}$. Note that, alternatively, we could also draw ratings from the empirical distribution of ratings for a group (so relaxing the Gaussian assumption) and also by generating user ratings via a water-filling approach i.e. split the data into training and test data, pick a user from the test data and use their ratings, when we need a rating for an item that the user has not rated, pick a second user from the same group who has rated the item and merge the pair of user ratings. We found the performance of these setups to be very similar to simply drawing a new user from a Gaussian distribution.

With user ratings $R(v_1), \ldots, R(v_t)$ and synthetic ratings $R(s_{t+2}), \ldots, R(s_{t_{max}})$ in hand, equation (1) can now be used to calculate the estimated probability $\{p_g^{(t_{max})}\}$ when the user rates items $\{v_1, \ldots, v_t, s_{t+2}, \ldots, s_{t_{max}}\}$. If the "true" group $\hat{g}$ is the group with highest estimated probability then the reward for the sequence is +1, else 0.

Averaging over multiple such samples, the average reward $Q(n)/N(n)$ will be

$$\frac{Q(n)}{N(n)} \approx \mathbb{E}[\text{Prob true group is correctly estimated} \mid \text{true group} = g, \text{user ratings of items } V^{(t)}, \text{ next items are } s_{t+2}, \ldots, s_{t_{max}}]$$

where the expectation is over the groups $\mathcal{G}$.

Algorithm 3 gives pseudo-code for this Monte Carlo-based reward calculation.

Algorithm 3 calculate_reward

```
Input: Item sequences $\mathcal{V}^{(t)}$, $\mathcal{S}$
Draw user group $\hat{g}$ according to probability distribution $\{p_g^{(t)}, g \in \mathcal{G}\}$
Using $\hat{g}$, generate synthetic user rating $R(s)$ for each item $s \in \mathcal{S}$
Calculate $p_g^{(t_{max})}, g \in \mathcal{G}$ using equation (1)
if $\hat{g} \in \arg\max_{g \in G} p_g^{(t_{max})}$ then
    return +1                ▷ Estimated group matches "true" group $\hat{g}$
else
    return 0
end if
```

3.2.4 Choosing The Next Item. A lookahead subtree consisting of sequences of items that tend to generate high reward is constructed by executing the Monte Carlo search steps (i)-(vi) *max_turns* times, where *max_turns* is a hyperparameter. We need to select *max_turns* to be large enough that promising sequences of items are discovered and are visited sufficiently frequently that the estimated reward for the sequence is reasonably accurate.

In the lookahead subtree the children of the tree root at the first items in the set of lookahead sequences. Using the lookahead subtree generated at step t we choose the next item v_{t+1} to ask the user to rate to be the child of the tree root that has been most visited during the Monte Carlo search i.e. $v_{t+1} = s(n^*)$ where $n^* \in \arg\max_{n \in children(\mathcal{T}_{root})} N(n)$. While we might alternatively choose the child with highest estimated reward $Q(n)/N(n)$ we found that selecting the most frequently visited child tended to achieve slightly better performance.

3.3 Further Speedups

3.3.1 Best Distinguisher Items Are Enough. The lookahead tree expands by a factor of roughly $|\mathcal{V}|$ at each level. Even with the Monte Carlo search approach described above the computational and memory burden can therefore quickly become substantial for large $|\mathcal{V}|$. Following [14], we note that some items are more effective than others for distinguishing between groups. For example, popular items rated highly by members of one group can tend to also be rated highly by members of at least some of the other groups, and so the user ratings for these items do not serve to strongly distinguish between groups.

Intuitively, an item v helps to distinguish whether a user belongs to group g rather than group h when (i) the mean rating of v by users in group g is very different from that of users in group h i.e. $(\mu(g, v) - \mu(h, v))^2$ is large, and (ii) when the ratings tend to be consistent/reliable i.e. the variance $\sigma^2(g, v)$ is small. That is, we expect that

$$\Gamma_{g,h}(v) = \frac{(\mu(g,v) - \mu(h,v))^2}{\sigma^2(g,v)}$$

is a measure of the ability of item v to distinguish group g from group h i.e. the larger $\Gamma_{g,h}(v)$ the better item v is at distinguishing group g from group h.

Another way to arrive at the same conclusion is to assume that for users belonging to group g the rating $R(v)$ of item v is i.i.d. gaussian with mean $\mu(g, v)$ and variance $\sigma^2(g, v)$. Consider similarity measure

$$R_t(g,h) = \sum_{i=1}^{t} \frac{R(v_i) - \mu(h, v_i)}{\mu(g, v_i) - \mu(h, v_i)}$$

Suppose the user belongs to group g. We expect that the deviations $R(v_i) - \mu(g, v_i), i = 1, \ldots, t$ tend to fluctuate around 0, as otherwise there would be a consistent offset between the user's ratings and the group ratings in which case the user would better be assigned to a different group. Therefore $R_t(g, h) \to 1$ as $t \to \infty$ for $h \neq g$ and, similarly, $R_t(h, g) \to 0$ as $t \to \infty$ for $h \neq g$. Hence, by thresholding $R_t(g, h)$ we can identify the user group g. By standard concentration inequalities, $Prob(|R_t(g,h) - 1| > \epsilon) < 2e^{-\frac{\epsilon^2}{2}\sum_{i=1}^{t}\Gamma_{1,h}(v_i)}$ and and $Prob(|R_n(h,g) - 0| > \epsilon) < 2e^{-\frac{\epsilon^2}{2}\sum_{i=1}^{t}\Gamma_{h,g}(v_i)}$. That is, for $R_t(g, h)$ to converge quickly we want $\sum_{i=1}^{t} \Gamma_{g,h}(v_i)$ to be large.

Using this observation, for each pair of groups $g, h \in \mathcal{G}$ we pick the t_{max} items v for which $\Gamma_{g,h}(v)$ is largest and add these to set $\hat{\mathcal{V}}$. Set $\hat{\mathcal{V}}$ is generally much smaller than set $\mathcal{V}$, e.g. Table 1 shows the size of $\hat{\mathcal{V}}$ for the standard Netflix dataset as the number of groups is varied. In our performance evaluation below we will select items to ask the user to rate from the subset $\hat{\mathcal{V}} \subset \mathcal{V}$ of distinguisher items rather than the full set of items $\mathcal{V}$.

	$\|\mathcal{V}\|$	$\|\hat{\mathcal{V}}\|$			
		4	**8**	**16**	**32**
Netflix	17,770	269	759	1,224	905
Jester	100	59	83	98	100

Table 1: Number of good distinguisher items vs #groups for the Netflix and Jester datasets.

3.3.2 No Rollouts. In Algorithm 1, after reaching a leaf node in the lookahead tree $\mathcal{T}$ of depth $j < t_{max}$ we select $j - t_{max}$ items at random to obtain a sample item sequence of length t_{max}. A *rollout* step of this sort is standard in MCTS. However, we also measured the performance of our MCTS-based approach when this rollout step was omitted i.e. a sample item sequence of length $j \leq t_{max}$ is generated from the lookahead tree.

Figure 4 shows typical performance measurements for the Netflix and Goodreads datasets. This plots the accuracy with which the user group is estimated vs the number of items t_{max} that are rated by the user. Data is shown both with and without the rollout step. It can be seen that, perhaps somewhat surprisingly, the rollout step tends to degrade performance. What we suspect is happening here is that the rollout step is effectively just adding unhelpful noise since we may only sample a relatively small number of random item sequences from the full set of possible sequences of length $j - t_{max}$, especially during the initial stages when j is small. Omitting the rollout step therefore offers the double advantage of reduced computation and improved performance.

3.3.3 Number Of Turns. The performance of the MCTS-based algorithm depends on the *max_turns* parameter that determines the number of Monte Carlo samples used to generate the lookahead subtree. This needs to be sufficiently large that promising sequences of items are discovered and are visited sufficiently frequently that the estimated reward for the sequence is reasonably accurate. Initially we used a simple linear heuristic to $max\_turns = k \times |G| \times |\hat{\mathcal{V}}|$, where parameter k is varied depending on the dataset but it typically around 200.

Dataset	k						
	1	*5*	*10*	*50*	*100*	*200*	*500*
Goodreads/8	0.564	0.608	0.615	0.659	0.679	0.710	0.703
Jester/8	0.485	0.491	0.530	0.524	0.536	0.545	0.551
Netflix/8	0.691	0.801	0.795	0.825	0.831	0.831	0.830

Table 2: Group estimation accuracy vs choice of *max_turns* parameter k. Netflix, Goodreads and Jester datasets, $|\mathcal{G}| = 8$ groups, $t_{max} = 5$ items rated by new user.

Table 2 shows the accuracy of the MCTS algorithm as the value of k is varied. Data is shown for the three datasets with $|\mathcal{G}| = 8$ groups and $t_{max} = 5$. It can be seen that the accuracy initially improves as k is increased. This is as expected since the lookahead substree is being more thoroughly explored. However, once k gets to a value of about 200 the accuracy starts to level off, indicating that the lookahead tree is now sufficiently large and accurate.

The value of k where the accuracy levels off depends on depth $d = t_{max} - t$ of the lookahead tree ($d = 5$ in Table 2), a smaller value of k being admissible when d is smaller, and vice versa. In our tests we therefore used $k = (1.25 + (t_{max} - t)^2$.

3.4 Software

Our MCTS-based cold start implementation and data is available on github at https://github.com/dilina-r/mcts-rec.

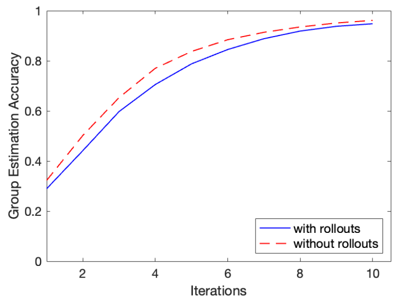
4 PERFORMANCE EVALUATION

4.1 Evaluation Setup

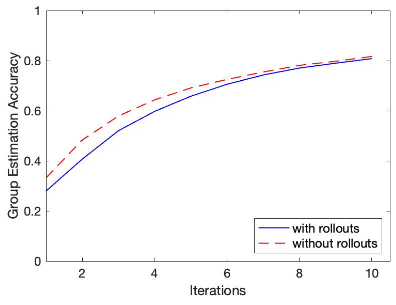
Datasets. We evaluate the performance of the cluster-based bandit algorithm for cold start on the standard Netflix dataset (480,189 people 17,770movies, 104M ratings from 1–5), the Jester dataset (73,421 people rating 100 jokes, 4.1M ratings from -10–10) and the Goodreads10K dataset (53,424 people rating10,000 books, 5.9M ratings from 1-5).

Clustering Users. We use training data to cluster users into groups and estimate the mean $\mu(g, v)$ and variance $\sigma(g, v)^2$ of the ratings by each group g for item v. We use the BLC matrix-factorization clustering algorithm [4] for this, although other clustering algorithms might also be used. We vary the number of groups/clusters from 4 to 32 and report results for each.

Baseline Algorithms. We compare the performance of the proposed MCTS-based approach against (i) an optimised CART decision tree and (ii) the cluster-based bandit (CBB) algorithm of [14]. These are strong baselines, with good performance for cold-start



(a) Netflix Dataset - 8 groups



(b) Goodreads Dataset - 8 groups

Figure 4: Group estimation accuracy with and without rollout step.

active learning. Decision-trees are often considered for use in cold-start while the recently proposed CB-algorithm offers state of the art performance [14].

Modelling New Users. We generate the item ratings of a new user from group g by making a single draw from the multivariate Gaussian distribution with mean $\mu(g, v)$ and variance $\sigma(g, v)^2$ for each item equal to that estimated from the training data. This has the advantage that we can easily generate large numbers of new users in a clean, reproducible manner. In addition, we also evaluated performance when drawing ratings by splitting the data into training and test data, picking a user from the test data and using their ratings. We found the performance of these setups to be very similar to simply drawing a new user from a Gaussian distribution.

Performance Metrics. We report the accuracy with which the group of a new user is estimated, i.e. the fraction of times the correct group is estimated, vs the number of items rated by a new user. Statistics are calculated over 1000 new users per group.

Hardware. Tests were carried out on an 8-Core Intel i7-9700 CPU @ 3.00GHz, with 8GB RAM. Computational performance was measured using only a single core of the CPU.

4.2 Results

Figure 5 shows measurements of the mean accuracy vs the number of items rated by a new user for the Netflix, Goodreads and Jester dataset with users clustered in 4 and 16 groups. Data is shown when using a decision tree (DT), the CBB algorithm of [14] and our proposed MCTS-based approach (UCT). To calculate the mean accuracy, 1000 new users are generated for each group and the accuracy averaged over the users and groups i.e. averaged over 4,000 users for 4 groups and 16,000 for 16 groups. It can be seen that the MCTS approach uniformly achieves a higher accuracy than the decision-tree and CBB approaches for a given number of items rated.

Figure 6 shows typical (i.e. representative of the full data) measurements of the per-group accuracy of the DT, CBB and MCTS approaches. It can seen that the MCTS approach consistently achieves higher accuracy for every group. The variation in the accuracy across groups is also lower, particularly in comparison to that of the decision-tree approach e.g. it can be seen in Figure 6(b) that the accuracy can range from about 30% to about 90& when using

dataset/#groups	*algorithm*	*#iterations* 5	*10*	*15*	*20*	*25*
Netflix/16	DT	0.359	0.194	0.196	0.196	0.196
	MCTS	**0.185**	**0.105**	**0.063**	**0.041**	**0.027**
Goodreads/16	DT	0.341	0.188	0.202	0.202	0.202
	MCTS	**0.216**	**0.173**	**0.137**	**0.113**	**0.094**
Jester/16	DT	**0.150**	0.146	0.154	0.154	0.154
	MCTS	0.182	**0.138**	**0.114**	**0.092**	**0.076**

Table 3: Standard deviation of the measured per-group accuracy for the decision tree (DT) and MCTS approaches. Netflix, Jester and Goodreads data with 16 groups.

the decision-tree. Table 3 shows the standard deviations of the per-group accuracies vs the number of items rated by a new user for the Netflix, Goodreads and Jester datasets with 16 groups. Once again, it can be seen that the variation in accuracy across groups is significantly lower with the MCTS approach.

Tables 4–6 summarise the measured group estimation accuracy vs the number of items rated by a new user and the number of user groups. Data is shown for 5–25 items and 4–32 groups and for the Netflix, Goodreads and Jester datasets. It can be seen that the MCTS-based approach achieves uniformly superior performance to the decision-tree and CBB approaches i.e. better performance regardless of the number of items the user rates, the number of user groups and the dataset used. The performance improvement is often considerable. For example, with 4 groups and asking the user to rate 5 items the MCTS accuracy on the Netflix dataset is 0.951 vs a decision-tree accuracy of 0.672 and a CBB accuracy of 0.705. Similarly, for 16 groups and the user rating 15 items the accuracies are 0.920, 0.631 and 0.854 for MCTS, DT and CBB respectively.

Since the performance of the MCTS approach dominates that of the other algorithms, this performance data indicates that it should always be used in preference to them if higher accuracy is desired.

4.3 MCTS Computation Time

Table 7 shows measurements of the average time taken to recommend the next item to a new user. This is the time taken at step t to construct a lookahead subtree containing item sequences that tend to generate high reward, this subtree is then used to select the next item to ask a new user to rate. The data shown makes use of a single

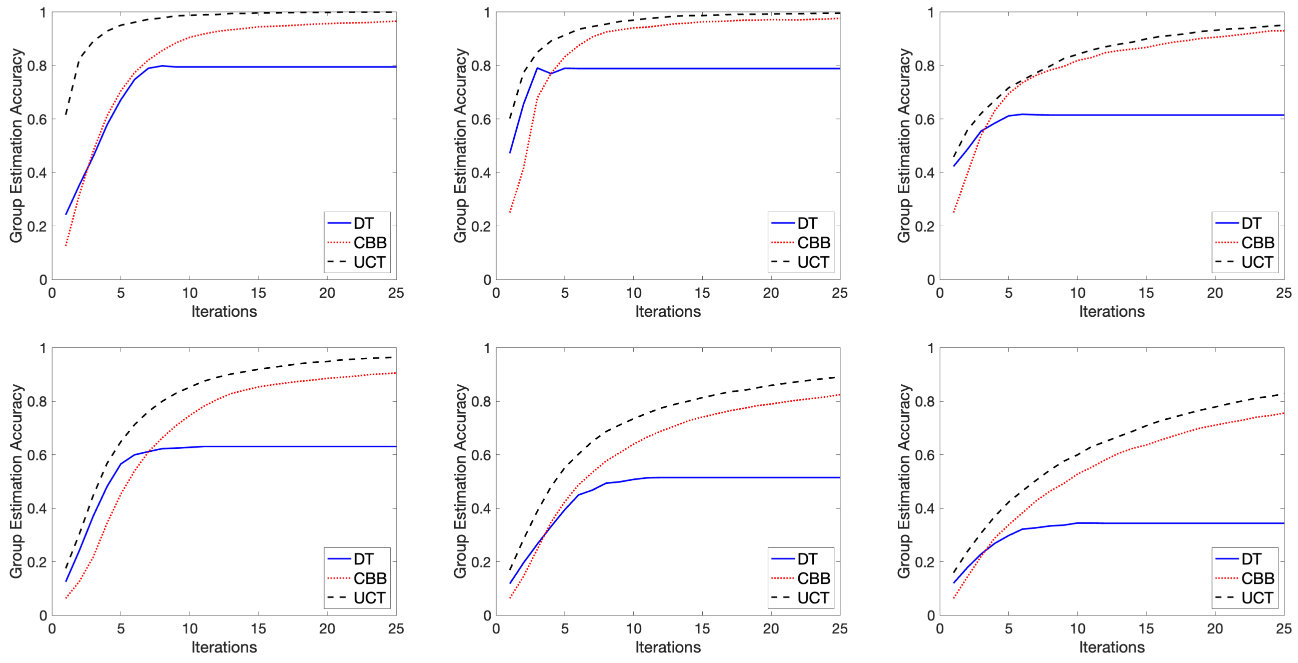
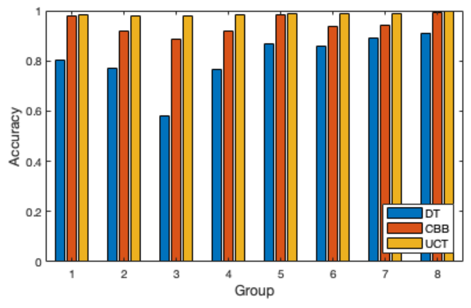
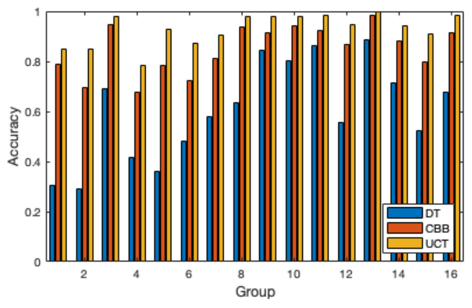


Figure 5: Measured mean estimation accuracy vs number of items rated by a new user. Data is shown for the decision tree (DT), CBB and MCTS-based approaches for the Netflix, Goodreads and Jester datasets with 4 and 16 groups



(a) 8 groups



(b) 16 groups

Figure 6: Measured per-group accuracy of the decision tree (DT), CBB and MCTS-based approaches. Netflix dataset with 8 and 16 groups, $t_{max} = 15$ items rated by new user.

CPU core. The MCTS algorithm is massively parallelisable in the sense that computations for different new users can be trivially run in parallel and so is linearly scalable with the number of CPU cores. The time taken depends on the lookahead tree expansion factor (i.e. the size of the set of items that a user can be asked to rate), hence why the time is somewhat longer for the Netflix dataset than for Goodreads. It also depends on the MCTS *max_turns* parameter that determines the number of Monte Carlo samples used to generate the lookahead subtree, shown in Table 7.

In summary, for 8 groups the MCTS-based approach takes around 30ms to recommend a item for a new user to rate, allowing over 30+ recommendations per second using a single commodity CPU core. Performance scales linearly with the number of CPU cores.

5 DISCUSSION

5.1 Including Contextual Information

Typically a recommender system will have some contextual information regarding a new user. For example, the country/city they are located in, user gender and demographics, data from other services used by the user. The proposed MCTS-based approach can be directly extended to incorporate such contextual information. For example, users sharing the same context (e.g. located the same country) can be gathered together, the number of user groups and the item rating means and variances for each group estimated and then the MCTS-based cold-startr approach applied to this sub-population.

#groups	*algorithm*	*#iterations*				
		5	*10*	*15*	*20*	*25*
4	DT	0.672	0.795	0.795	0.795	0.795
	CBB	0.705	0.906	0.945	0.957	0.966
	MCTS	**0.951**	**0.988**	**0.997**	**0.999**	**1.000**
8	DT	0.674	0.780	0.780	0.780	0.780
	CBB	0.701	0.903	0.942	0.955	0.964
	MCTS	**0.838**	**0.962**	**0.987**	**0.994**	**0.997**
16	DT	0.566	0.628	0.631	0.631	0.631
	CBB	0.453	0.747	0.854	0.886	0.906
	MCTS	**0.649**	**0.853**	**0.920**	**0.949**	**0.965**
32	DT	0.327	0.414	0.425	0.423	0.423
	CBB	0.272	0.517	0.651	0.715	0.755
	MCTS	**0.437**	**0.642**	**0.739**	**0.797**	**0.836**

Table 4: Mean estimation accuracy vs the number of items rated by a new user and the number of user groups. Netflix movie dataset

#groups	*algorithm*	*#iterations*				
		5	*10*	*15*	*20*	*25*
4	DT	0.790	0.789	0.789	0.789	0.789
	CBB	0.833	0.941	0.964	0.972	0.977
	MCTS	**0.914**	**0.970**	**0.987**	**0.993**	**0.996**
8	DT	0.582	0.596	0.594	0.594	0.594
	CBB	0.593	0.774	0.817	0.851	0.874
	MCTS	**0.691**	**0.817**	**0.870**	**0.903**	**0.928**
16	DT	0.395	0.508	0.515	0.515	0.515
	CBB	0.425	0.640	0.741	0.790	0.825
	MCTS	**0.552**	**0.735**	**0.814**	**0.860**	**0.891**
32	DT	0.247	0.310	0.304	0.303	0.303
	CBB	0.269	0.448	0.548	0.609	0.656
	MCTS	**0.368**	**0.544**	**0.647**	**0.708**	**0.758**

Table 5: Mean estimation accuracy vs the number of items rated by a new user and the number of user groups. Goodreads books dataset.

#groups	*algorithm*	*#iterations*				
		5	*10*	*15*	*20*	*25*
4	DT	0.612	0.615	0.615	0.615	0.615
	CBB	0.695	0.819	0.868	0.906	0.930
	MCTS	**0.717**	**0.842**	**0.900**	**0.932**	**0.951**
8	DT	0.415	0.463	0.463	0.463	0.463
	CBB	0.484	0.670	0.750	0.801	0.837
	MCTS	**0.542**	**0.701**	**0.790**	**0.847**	**0.883**
16	DT	0.298	0.345	0.344	0.344	0.344
	CBB	0.338	0.528	0.637	0.711	0.756
	MCTS	**0.423**	**0.600**	**0.709**	**0.779**	**0.828**
32	DT	0.206	0.244	0.248	0.249	0.249
	CBB	0.228	0.410	0.530	0.615	0.673
	MCTS	**0.305**	**0.494**	**0.613**	**0.701**	**0.761**

Table 6: Mean estimation accuracy vs the number of items rated by a new user and the number of user groups. Jester jokes dataset.

Dataset	**#groups**	**max_turns**	**avg. time**
Netflix	8	193312	30.90ms
	16	349184	86.10ms
Goodreads	8	151896	23.80ms
	16	270528	46.78ms
Jester	8	79376	15.14ms
	16	166400	19.17ms

Table 7: Mean MCTS computation time to recommend next item to a new user.

This is similar to the approach used in contextual bandits. Contextual information can also be used to adjust the prior probabilities $p_g^{(0)}, g \in \mathcal{G}$ of the group to which the user is initially estimated to belong.

5.2 Other Types of Feedback

In this paper we assume that new users provide item ratings as feedback, but our approach can be readily extended to encompass other types of user feedback. For example, a new user might be presented with two items and asked which they prefer. In the MCTS-based approach the nodes in the lookahead tree now correspond to pairs of items but the approach is otherwise unchanged in principle. It is necessary to be able to estimate the group probabilities $p_g^{(t)}, g \in \mathcal{G}$ from user feedback up to time t and to generate synthetic user feedback for the Monte Carlo step in the MCTS algorithm, but standard models can be used for this e.g. the Bradley-Terry model for paired comparisons

5.3 Offline Training

In the approach considered here a new lookahead subtree is constructed at each step t in order to select the next item to ask a new user to rate. While our measurements show that this tree can be constructed rather quickly, a further computational saving might be possible by storing the generated lookahead subtrees for new users and using these as training data for a neural net. That is, it might be possible to train a neural net offline to capture the decisions made by the MCTS approach and then use this neural net to make online predictions. However, we leave investigation of this interesting topic to future work.

6 CONCLUSIONS

In this paper we revisit the cold-start task for new users of a recommender system whereby a new user is asked to rate a few items with the aim of discovering the user's preferences. We propose using a Monte Carlo Tree Search (MCTS) based approach to dynamically select the sequence of items presented to a new user. We find that this new MCTS-based cold-start approach is able to consistently quickly identify the preferences of a user with significantly higher accuracy than with either a decision-tree or a state of the art bandit-based approach without incurring higher regret i.e the learning performance is fundamentally superior to that of the state of the art. This boost in recommender accuracy is achieved in a computationally lightweight fashion.

ACKNOWLEDGMENTS

This work was supported by Science Foundation Ireland (SFI) under grant 16/IA/4610

REFERENCES

[1] Xavier Amatriain, Neal Lathia, Josep M. Pujol, Haewoon Kwak, and Nuria Oliver. 2009. The Wisdom of the Few: A Collaborative Filtering Approach Based on Expert Opinions from the Web. In *Proc SIGIR* (Boston, MA, USA). 532–539. https://doi.org/10.1145/1571941.1572033

[2] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[3] Rocio Canamares, Marcos Redondo, and Pablo Castells. 2019. Multi-Armed Recommender System Bandit Ensembles. In *Proc RecSys*. https://doi.org/10.1145/3298689.3346984

[4] Alessandro Checco, Giuseppe Bianchi, and Douglas J. Leith. 2017. BLC: Private Matrix Factorization Recommenders via Automatic Group Learning. *ACM Trans. Priv. Secur.* 20, 2, Article 4 (May 2017), 25 pages. https://doi.org/10.1145/3041760

[5] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*. Springer, 72–83.

[6] Mattia Crippa, Pier Luca Lanzi, and Fabio Marocchi. 2022. An analysis of Single-Player Monte Carlo Tree Search performance in Sokoban. *Expert Systems with Applications* 192 (2022), 116224.

[7] Mehdi Elahi, Matthias Braunhofer, Tural Gurbanov, and Francesco Ricci. 2018. User Preference Elicitation, Rating Sparsity and Cold Start. *Collaborative Recommendations* (2018), 253–294. https://doi.org/10.1142/9789813275355_0008

[8] Mehdi Elahi, Francesco Ricci, and Neil Rubens. 2016. A survey of active learning in collaborative filtering recommender systems. *Computer Science Review* 20 (2016), 29–50. https://doi.org/10.1016/j.cosrev.2016.05.002

[9] Crícia Z. Felício, Klérisson V.R. Paixão, Celia A.Z. Barcelos, and Philippe Preux. 2017. A Multi-Armed Bandit Model Selection for Cold-Start User Recommendation. In *Proc UMAP* (Bratislava, Slovakia). 32–40. https://doi.org/10.1145/3079628.3079681

[10] Nadav Golbandi, Yehuda Koren, and Ronny Lempel. 2011. Adaptive Bootstrapping of Recommender Systems Using Decision Trees. In *Proc WSDM* (Hong Kong, China). 595–604. https://doi.org/10.1145/1935826.1935910

[11] Al Mamunur Rashid, George Karypis, and John Riedl. 2008. Learning Preferences of New Users in Recommender Systems: An Information Theoretic Approach. *SIGKDD Explor. Newsl.* 10, 2 (Dec. 2008), 90–100. https://doi.org/10.1145/1540276.1540302

[12] Maarten PD Schadd, Mark HM Winands, HJVD Herik, Guillaume MJ-B Chaslot, and Jos WHM Uiterwijk. 2008. Single-player monte-carlo tree search. In *International Conference on Computers and Games*. Springer, 1–12.

[13] Maarten PD Schadd, Mark HM Winands, Mandy JW Tak, and Jos WHM Uiterwijk. 2012. Single-player Monte-Carlo tree search for SameGame. *Knowledge-Based Systems* 34 (2012), 3–11.

[14] Sulthana Shams, Daron Anderson, and Douglas Leith. 2021. Cluster-Based Bandits: Fast Cold-Start for Recommender System New Users. In *Proc SIGIR*.

[15] Lei Shi, Wayne Xin Zhao, and Yi-Dong Shen. 2017. Local Representative-Based Matrix Factorization for Cold-Start Recommendation. *ACM Trans. Inf. Syst.* 36, 2, Article 22 (Aug. 2017), 28 pages. https://doi.org/10.1145/3108148

[16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[18] Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. 2011. Functional Matrix Factorizations for Cold-Start Recommendation. In *Proc SIGIR*. 315–324. https://doi.org/10.1145/2009916.2009961