Preference Elicitation Algorithms

Vagul Mahadevan, Declan Stacy, and Nathan Weatherly

April 2023

1 Introduction

In this report we experiment with some algorithms for preference elicitation in interactive recommender systems. We have some set $I = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$ of items and a single user with (unknown) preferences. Their utility is given by $u(x) = \theta_*^T x$, where $\theta_* \in \mathbb{R}^d$. The goal is to figure out $x_* = \text{argmax}_{1 \leq i \leq n} u(x_i)$. We are allowed to ask the user questions like "what is your utility for item x" or "what is your preferred item from the set $\{x_1, \ldots, x_K\}$," and we wish to find x_* in the least number of interactions possible. We will tackle this problem in three different settings, and in each one we will come up with a solution, implement our algorithm, and run simulations to report the performance of our algorithm.

Throughout this report, we will assume that the set of items I is generated by iid samples from a continuous distribution on $\mathbb{R}^d$. In our simulations, we choose them to uniformly distributed on the sphere $S^{d-1} = \{x \in \mathbb{R}^d \mid ||x|| = 1\}$ (where $||-||$ always denotes the L^2 norm). We may assume that θ_* satisfies $||\theta_*|| = 1$, as scaling the user's utility function does not affect their preferences. We will also assume $n \geq d$ (n is the number of items). Unless otherwise noted, all vectors will be column vectors.

2 Part 1

2.1 Setting

In each round t we choose an item x_t and receive the user's exact utility $u(x_t)$. After some number of interactions (rounds) T, we pick an item x, and we want $x = x_*$. We also wish to minimize T.

2.2 Solution Explanation

By asking the user to provide their utility of d items, we can calculate θ_* exactly. This is because, for each item x_i, we have $\theta_*^\top x_i = y_i$, and we can form a d by $d+1$ augmented matrix to represent this system of equations (d equations in d unknowns). Because the x_i are all drawn independently from a continuous distribution, the probability that they will be linearly dependent is 0, and so we

can use row reduction to solve the system of equations and find an exact value for θ_*. Then, we can loop through I and choose the item that maximizes $\theta_*^\top x_i$. This will find the best item in exactly d interactions, regardless of the number of items.

2.3 Algorithm Pseudocode

1. Form a d by d matrix A whose rows consist of $x_1^T, \ldots, x_d^T$, the first d items in I
2. For $1 \leq i \leq d$, get the users utility y_i for item i and store them in a vector y
3. Let θ be the solution to $A\theta = y$
4. Return $\text{argmax}_{x \in I}\, \theta^T x$

2.4 Results

The algorithm worked as intended; every time it finds the best item in exactly d iterations.

3 Part 2

3.1 Setting

In each round t we choose an item x_t and get the user's (noisy) response $y_t = u(x_t) + \epsilon_t$, where the ϵ_t are iid with distribution $N(0, \sigma^2)$ (for our simulations, we use $\sigma = 0.1$). After some number of interactions (rounds) T, we pick an item x, and we wish to maximize $u(x)$ (equivalently, minimize $u(x_*) - u(x)$) while minimizing the number of interactions T.

3.2 Solution Explanation

Our situation after round t can be summarized as $y = A\theta_* + z$, where y is the column vector of noisy utilities y_i, A is a $t \times d$ matrix with rows x_i^T, and $z \sim N(0, \sigma^2 I)$, where I denotes the $t \times t$ identity matrix. We have no knowledge of z or θ_*, only y and A.

At each step we are going to run linear regression to find the best guess θ which minimizes $||A\theta - y||$. Thus, $\theta = (A^T A)^{-1} A^T y$. Let $x = \text{argmax}_{1 \leq i \leq t}\, \theta^T x_i$, our best guess for x_*. In order to run linear regression in the first place, we need A to have at least d rows so that $A^T A$ is invertible. The first task is thus to choose the first d items to add to A. We first do some calculations regarding the distribution of some key quantities we are concerned with, and then use these to develop a heuristic for selecting the first d rows of A.

3.2.1 Motivation and Preliminary Calculations

We can explicitly calculate the distribution of

$$\theta = (A^TA)^{-1}A^Ty = (A^TA)^{-1}A^T(A\theta_* + z) = \theta_* + (A^TA)^{-1}A^Tz$$

as $N(\theta_*, (A^TA)^{-1}A^T(\sigma^2 I)A(A^TA)^{-1}) = N(\theta_*, \sigma^2(A^TA)^{-1})$.

Then for any x, $u(x) - \theta^T x = x^T(\theta_* - \theta)$ has distribution $N(0, \sigma^2 x^T(A^TA)^{-1}x)$. Thus,

$$u(x_*) - u(x) = u(x_* - x) - \theta^T(x_* - x) + \theta^T(x_* - x)$$

has distribution

$$N(\theta^T(x_* - x), \sigma^2(x_* - x)^T(A^TA)^{-1}(x_* - x))$$

When $|(A^TA)^{-1}| \coloneqq \max_{||v||=1} v^T(A^TA)^{-1}v$ is small enough, meaning the smallest eigenvalue of A^TA is large enough, we can guarantee with some high probability α that $u(x_*) - u(x)$ is smaller than some small ϵ (since $\text{Var}[u(x_*) - u(x)] \leq \sigma^2||x_* - x||^2|(A^TA)^{-1}|$.) So we can start off by choosing the items in each round so that the smallest eigenvalue of A^TA is made as large as possible. If we had an orthonormal eigenbasis $v_1, \ldots, v_d$ for A^TA with associated eigenvalues $0 \leq \lambda_1 \cdots \leq \lambda_d$, we could choose our next item x_{t+1} to be the one which maximizes $v_1^T x_{t+1}$, as this would increase the lowest eigenvalue to at least $\min(\lambda_1 + (v_1^T x_{t+1})^2, \lambda_2)$.

To see this, note that adding a row x_{t+1}^T to A changes A^TA by adding $x_{t+1}x_{t+1}^T$ and also that the smallest eigenvalue of a symmetric matrix M is given by $\min_{||v||=1} v^TMv$. Expressing A^TA as $\Sigma D\Sigma^T$ where Σ has columns v_i and D is diagonal with entries λ_i,

$$\begin{aligned}
\min_{||v||=1} v^T(A^TA + x_{t+1}x_{t+1}^T)v &= v^T\Sigma D\Sigma^T v + v^T x_{t+1}x_{t+1}^T v \\
&\geq \lambda_1(v^Tv_1)^2 + \lambda_2(1 - (v^Tv_1)^2) + (v^Tx_{t+1})^2 \\
&= \lambda_1(v^Tv_1)^2 + \lambda_2(1 - (v^Tv_1)^2) + (v^T\Sigma\Sigma^Tx_{t+1})^2 \\
&\geq \lambda_1(v^Tv_1)^2 + \lambda_2(1 - (v^Tv_1)^2) + (v^Tv_1v_1^Tx_{t+1})^2 \\
&= \lambda_2 + (v^Tv_1)^2(\lambda_1 + (v_1^Tx_{t+1})^2 - \lambda_2) \\
&\geq \min(\lambda_1 + (v_1^Tx_{t+1})^2, \lambda_2)
\end{aligned}$$

3.2.2 Initialization of A

That was the motivation for the initialization of our algorithm, but we do not want to actually keep track of an orthonormal eigenbasis and eigenvalues. Luckily, we don't have to because we need only know the span of the eigenspace corresponding to the eigenvalue 0, which by inspection is exactly the perp of the space spanned by the rows of A. ($A^TA = \sum_i x_i x_i^T$ where x_i^T is the ith row of A.) Thus, we iteratively choose the ith row x_i^T to maximize $||x_i||^2 - \sum_{j<i}(x_i^T\hat{x_j})^2$, where $\hat{x}_1, \ldots, \hat{x}_{i-1}$ is an orthonormal basis for $\text{Span}(x_1, \ldots, x_{i-1})$. Geometrically this amounts to choosing items which are as close to perpendicular from the other items as possible.

3.2.3 Adaptive Part of Algorithm

Now that we have initialized A, we can calculate θ and x for the first time. From here on out, we can use these quantities as the basis for our subsequent decisions. Instead of minimizing $|(A^TA)^{-1}| \coloneqq \max_{||v||=1} v^T(A^TA)^{-1}v$, we will focus on minimizing $v^T(A^TA)^{-1}v$ only for the specific values of v which have the largest effect on our error.

Since our entire goal is to minimize $u(x_*) - u(x)$, we choose some confidence level α and for each $x_i \neq x$ compute e_i such that $P(u(x_i) - u(x) < e_i) = \alpha$. Then with probability α our error is at most $e = \max e_i$ no matter what x_* is. Using our expression for the distribution of $u(x_i) - u(x)$ from the previous section, we calculate

$$e_i = \theta^T(x_i - x) + \beta\sigma\sqrt{(x_i - x)^T(A^TA)^{-1}(x_i - x)}$$

for some number of standard deviations β. If $x' \neq x$ is the item with the highest value of e (the one we are the least sure about), we would like to decrease $(x' - x)^T(A^TA)^{-1}(x' - x)$ in the next iteration.

(Note: This is extremely similar to linear UCB, but instead of getting an upper confidence bound on $u(x_i)$, we get an upper confidence bound on our error $u(x_i) - u(x)$.)

By Sherman-Morrison Formula, if $B = (A^TA)^{-1}$ and $\hat{A}$ is the result of adding a row v^T to A, then $\hat{B} \coloneqq (\hat{A}^T\hat{A})^{-1} = B - \frac{Bvv^TB}{1+v^TBv}$. If we want to minimize $x^T\hat{B}x = x^TBx - \frac{x^TBvv^TBx}{1+v^TBv} = x^TBx - \frac{(v^TBx)^2}{1+v^TBv}$, that is equivalent to maximizing $\frac{(v^TBx)^2}{1+v^TBv}$. So after finding the x' which maximizes $e_{x'} = \theta^T(x' - x) + \beta\sigma\sqrt{(x' - x)^TB(x' - x)}$, the algorithm should choose the next item x'' to maximize $\frac{(x''^TB(x'-x))^2}{1+x''^TBx''}$.

3.3 Algorithm Pseudocode

In total, we have the following algorithm with parameters ϵ (how close we want to be) and β (representing how many standard deviations to use for calculating e). We also impose a maximum number of iterations max_iter to guarantee that the number of interactions $T \leq$ max_iter:

1. Choose items $x_1, \ldots, x_d$ iteratively, where at each step i, x_i is chosen to maximize $||x_i||^2 - \sum_{j<i}(x_i^T\hat{x_j})^2$ and $\hat{x_i} \coloneqq \frac{x_i - \sum_{j<i} x_i^T\hat{x}_j}{||x_i - \sum_{j<i} x_i^T\hat{x}_j||}$

2. Initialize a $d \times d$ matrix A with rows $x_1^T, \ldots, x_d^T$ and let $C = A^TA$ and $B = C^{-1}$

3. Get the users utility $y_1, \ldots, y_d$ for the d items chosen. Initialize a column vector y consisting of the y_is.

4. Initialize $e = \infty$ and num_iter $= d$

5. While $e > \epsilon$ and num_iter $<$ max_iter, do the following:

6. (a) Update $\theta = BA^T y$
   (b) Compute $x = \text{argmax}_{x \in I} \, \theta^T x$
   (c) For $x' \in I, x' \neq x$, compute $e_{x'} = \theta^T(x' - x) + \beta\sigma\sqrt{(x' - x)^T B(x' - x)}$
   (d) Update $e = \max_{x' \in I, x' \neq x} e_{x'}$ and let $x' = \text{argmax}_{x' \in I, x' \neq x} \, e_{x'}$
   (e) Find $x'' = \text{argmax}_{x'' \in I} \, \frac{(x''^T B(x' - x))^2}{1 + x''^T B x''}$
   (f) Get the user's utility y' for item x''.
   (g) Update y by adding y' as an entry.
   (h) Update A by adding x''^T as a row.
   (i) Update C by adding $x'' x''^T$ to it.
   (j) Update B by subtracting $\frac{B x'' x''^T B}{1 + x''^T B x''}$
   (k) Update num_iter by adding 1

7. Return x

3.4 Results

To begin, we tested 1000 trials of $n = 500$ items and dimension $d = 20$. As a benchmark, we ran a naive algorithm which samples every single item (so $T = 500$) and chooses the item with the highest reported noisy utility. We also tested the algorithm above with the modification that the first d items are chosen randomly (version 0 .) The algorithm as described above is labeled as version 2. We use $\epsilon = .1$ and $\beta = .2$ and had the algorithm stop after $T = 40$ regardless of whether or not $e \leq \epsilon$. The average value of T was 33.33 for version 0 and 29.94 for version 2, an improvement of 10%. As another control, we also tested the algorithm from part 1 ($T = d = 20$), but the average suboptimality gap was so large that it did not fit on the graph (.428), where the suboptimality gap is defined as $u(x) - u(x_*)$ (recall that x is the item selected by the algorithm and x_* is the true best item). Version 2's average suboptimality gap was about 20% better than version 0's (0.0219 versus 0.0274), and both were a lot better than the naive's average of 0.0659.

We also tested $n = 1000$ and $d = 50$ for 500 trials, with the same parameters except a maximum number of interactions of 80 instead of 40. Version 0 had an average suboptimality of 0.0579 and version 2 (the actual algorithm) had 0.0421 (27% better), smashing the naive's 0.0824 and part 1's 0.4111. The average number of iterations was 78.612 for version 0 and 75.532 for version 2.

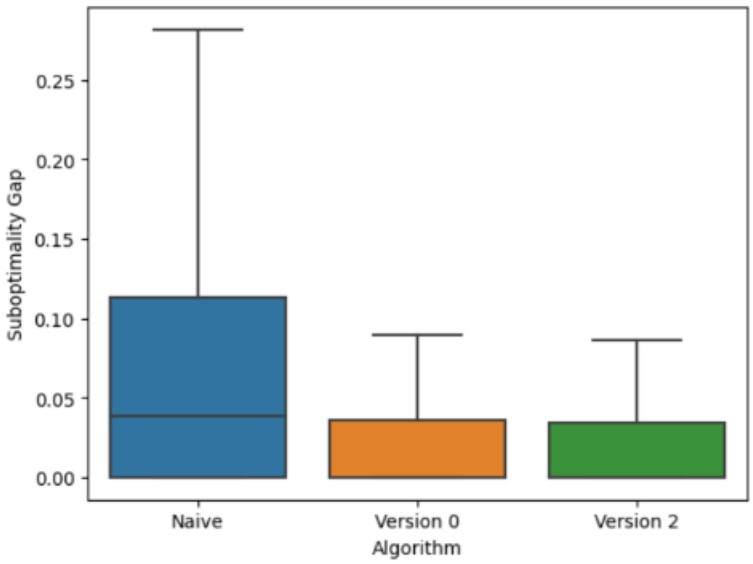

Figure 1: Results for $n = 500$, $d = 20$

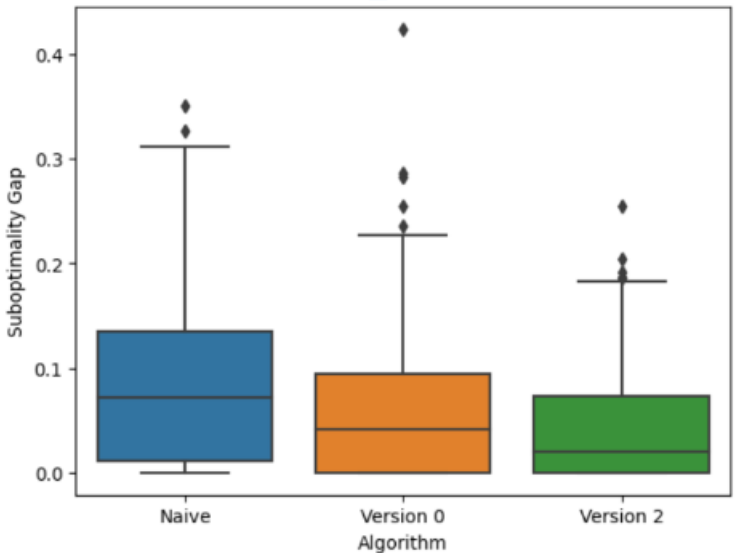

Figure 2: Results for $n = 1000$, $d = 50$

4 Parts 3/4

4.1 Setting

In this setting we do not observe the user's utility for any item. Instead, in each round t we present the user with a set $\{x_{1,t}, \ldots, x_{K,t}\}$ of K items, where $K \geq 2$ is a fixed constant, and the user responds with $x_{i,t}$, their most preferred item from the set. After some number of interactions (rounds) T, we pick an item x, and we wish to maximize $u(x)$ (equivalently, minimize $u(x_*) - u(x)$) while minimizing the number of interactions T.

4.2 Solution Explanation

4.2.1 Motivation

In each round t we are given information in the form of $\theta_*^T v_{j,t} \geq 0$ for vectors $v_{j,t} = x_{i,t} - x_{i+j,t}$ (where indices are taken modulo K to be in the range $1, \ldots, K$). We can maintain a set V consisting of all these vectors $v_{j,t}$ and let S be the positive linear span of V. Thus, S consists of all of the vectors v which we know satisfy $\theta_*^T v \geq 0$. Additionally, we keep track of our best guess θ for θ_*, which we will normalize to satisfy $||\theta|| = 1$. The goal is to continue until we are guaranteed that the angle between θ and θ_* is small, or in other words that $\theta^T \theta_*$ is big. (Recall that the angle between two vectors v and w is given by $\frac{\arccos(v^T w)}{||v||||w||}$.) If we choose θ such that the set of all x within some (hopefully large) angle a of θ lies within S, then we are guaranteed that θ_* is within $90 - a$ degrees of θ (otherwise the vector v which is a degrees away from θ on the opposite side of θ_* in the plane spanned by θ and θ_* is more than 90 degrees away from θ_*, so $\theta_*^T v \leq 0$, but $v \in S$, a contradiction.)

Thus, we want to find a θ which is sort of "in the middle" of S. Precisely, we want to find $\theta \in S$ which maximizes the minimum angle between θ and any point on the boundary of S. One possible approach to solve this is to sample a bunch of points P on the boundary of $S' = \{x \in S \mid ||x|| = 1\}$ and choose $\theta \in S'$ to minimize $\max_{p \in P} p^T \theta$. There are a few challenges with this optimization problem. The first is that S' is not convex. The second is that it requires us to keep track of which points are on the boundary of S and which points are in the interior.

4.2.2 SVM Approach

Instead, we opt for an easier approach which is intuitively quite similar and yields good results. Visually one can imagine that if we chose θ "in the middle" of S, the hyperplane θ^T would provide a nice bit of separation between S and $-S = \{-x \mid x \in S\}$. Thus, to find θ we treat our situation as a classification problem where we want a linear classifier which labels the points in S as positive and the points in $-S$ as negative. This is a perfect setup to use Support Vector Machine, which we can easily implement using pre-existing packages. In fact, the

separable case of SVM, assuming an intercept of 0 since our data is symmetric, is the optimization problem to maximize $\min_{p \in P} \frac{p^T \theta}{||\theta||}$, or equivalently to maximize $\min_{p \in P} p^T \theta$ subject to $||\theta|| = 1$. This is essentially the same problem we had formulated in the previous paragraph but with the mins and maxes swapped.

Another advantage of using SVM is that, provided we normalize the vectors in V, it allows us not worry about which points are on the boundary of S versus the interior, since any point $p \in S \backslash V$ will always satisfy $\frac{p^T \theta}{||p||} > \min_{p \in V} p^T \theta$. Intuitively, this is because points on the interior, when normalized to have norm 1, will always be further away from the hyperplane $\theta^\perp$ than points on the boundary. Mathematically this is a simple consequence of the convexity of $|| - ||$. Remember that S consists of positive linear combinations of vectors in V, which are exactly scaled versions of convex combinations of vectors in V. If $p \in S \backslash V$, then $p = C \sum t_i v_i$ for some $v_i \in V$, $C > 0$, and $t_i \in [0, 1]$ satisfying $\sum t_i = 1$, which means that

$$\frac{p^T \theta}{||p||} = \frac{C \sum t_i v_i^T \theta}{||C \sum t_i v_i||} \geq \frac{\sum t_i \min_{p \in V} p^T \theta}{|| \sum t_i v_i||} \geq \frac{\min_{p \in V} p^T \theta}{\sum t_i ||v_i||} = \min_{p \in V} p^T \theta$$

To summarize, it suffices to choose our set of sampled points P to just be V, as adding more points will not affect the performance.

Now that we have calculated our guess θ, we need to choose the next K vectors to ask the user about. Knowing the sign of $\theta_*^T v$ is most helpful when $\theta^T v$ is close to 0, as this is a value of v we are very uncertain about classifying. Thus, we want to choose items $x_{1,t}, \ldots, x_{K,t}$ such that $|\theta^T (x_{i,t} - x_{j,t})|$ is small for all $i \neq j$. This is easy; we sort the items x_i based on their estimated utility $\theta^T x_i$ and choose the continuous sublist of K items which minimizes $|\theta^T x_i - \theta^T x_{i+K-1}|$.

4.3 Algorithm Pseudocode

All of this can be summarized by the following algorithm:

1. Initialize a list of items V to be empty
2. Choose the first K items $x_1, \ldots, x_K$ and ask the user for their preferred item x_i.
3. For $1 \leq j < K$, let $x = x_i - x_{i+j}$ and add $\frac{x}{||x||}$ to V
4. Run a separable case of SVM with the points in L labeled as positive and the points in $-L$ labeled as negative to get a linear classifier $\theta^T x \geq 0$
5. For each item x, compute $\theta^T x$ and store in a list L
6. Sort L and find the index i which minimizes $L[i + K - 1] - L[i]$
7. Go back to step 2 using items corresponding to the K elements $L[i], \ldots, L[i + K - 1]$
8. After max_iter iterations, return $\text{argmax}_{x \in I} \theta^T x$

4.4 Results

We tested our algorithm against a naive algorithm that kept track of the best item observed so far and compared it with $K-1$ new items at each step. Below we graph the average best utility found against the number of interactions with the user. We ran 100 trials with $n = 1000$, $d = 20$, $K = 2$, and max_iter $= 300$. We also ran 500 trials with $n = 1000$, $d = 20$, $K = 5$, and max_iter $= 50$.

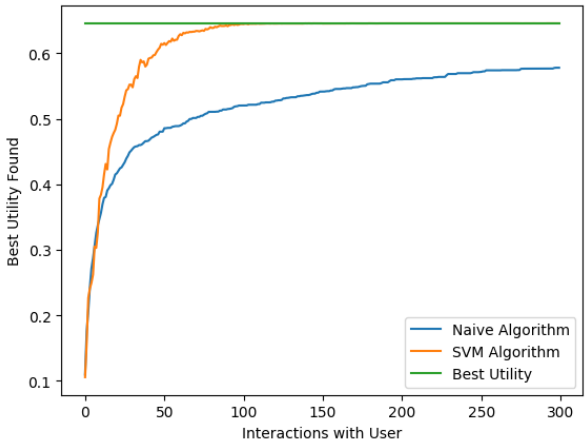

Figure 3: Results for $n = 1000$, $d = 20$, $K = 2$

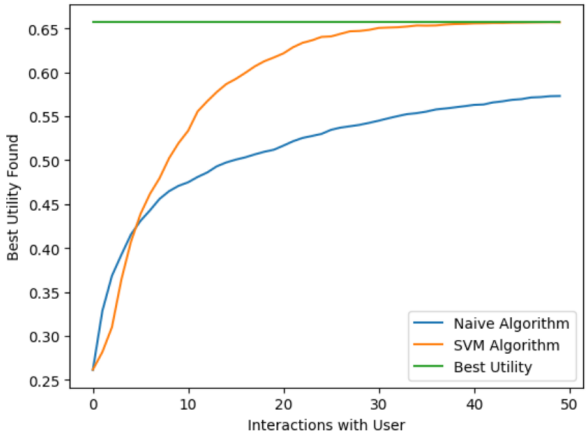

Figure 4: Results for $n = 1000$, $d = 20$, $K = 5$

5 Final Notes

Our algorithms worked as intended, but there is definitely room for improvement. For example, for the last algorithm we just stopped after a certain number of interactions to graph its performance as the number of interactions increased, but in practice a smarter stopping rule such as stopping after the guess θ has not changed much across the past few iterations may be better. Despite optimizations made in the code to make the algorithms from sections 3 and 4 faster, they are still quite slow, and perhaps for large values of n and d approximate versions of them which do less computation may work out better. That being said, our goal was to minimize the number of interactions with the user, not necessarily provide the fastest algorithm. The work done in section 3 to derive a rule for intelligently choosing the first d items did have a significant impact on the performance of the algorithm, as demonstrated by "version 2" outperforming "version 0" by 20 to 25%. We also at first experimented with an algorithm even more similar to linear UCB which chooses the next item x'' to be x' (see the pseudocode in section 3), but this also did worse than our algorithm. As an extension to this project, we would be curious to see how much the performance of section 4's algorithm would be improved by finding a way to effectively implement the ideas in section 4.2.1, which was our original motivation for the SVM-based algorithm we ended up creating.