COMP4121 Lecture Notes - Topic 3

More Fixed Point Algorithms: Online Voting and Ranking

LiC: Aleks Ignjatovic

ignjat@cse.unsw.edu.au

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Please read Chapters 5 and 6 in the textbook **Networked Life**, entitled *"When can I trust an average rating on Amazon?"* and *"Why Does Wikipedia even work?"*.

Voting: aggregation of individual preferences into a voting outcome

Voting can take many forms in terms of both the format of expressing individual preferences and in terms of how these individual preferences are aggregated into a voting outcome. Most common voting schemas are **positional voting** schemas. Individual preferences are expressed by voters providing rank ordered lists of (some of the) candidates. Vote Aggregation can be done in several ways:

1. **Plurality Voting:** We count the number of times each candidate was put on the first place; the highest scoring candidate is put on the first place on the final list, then the second, etc.
2. **Borda Count:** If there are M candidates, then the first placed candidate on a list gets $M-1$ points, the second $M-2$ points and so on; the last candidate gets no points. We now rank candidates according their total number of points across the lists of all voters.
3. **Condorcet Voting:** We compare every pair of candidates A, B; if more voters placed A above B then A is placed above B in the final list. Note that this method is not guaranteed to produce a consistent output list. For example, if there are three candidates A, B, C and three voters with the following personal preferences; voter 1: $A > B > C$; voter 2: $B > C > A$; voter 3: $C > A > B$. Then A should be placed above B; B should be placed above C and C should be placed above A which is clearly impossible.

How consistent are the above three voting schemata? Assume that we have $M = 3$ candidates, A, B and C and $N = 9$ voters, who voted as follows:

1. $A > B > C$ – 4 voters;
2. $C > B > A$ – 3 voters;
3. $B > C > A$ – 2 voters.

Then according to the plurality vote the ordering should be $A > C > B$ and thus A wins; according to the Borda count, A gets $2 \times 4 = 8$; B gets $1 \times 4 + 1 \times 3 + 2 \times 2 = 11$ and C gets $2 \times 3 + 1 \times 2 = 8$, so B wins and A and C tie; finally according to the Condorcet voting, B should be above A, B should be above C and C above A, thus producing the final outcome $B > C > A$, i.e., B wins, but, as we saw, Condorcet voting might fail to produce a consistent outcome.

Homework Problem 5. (a) Find the minimal number of voters and their individual voting preferences for 3 candidates A, B and C such that the Plurality Voting proclaims A as the winner, the Borda Count makes B the winner and the Condorcet Voting is consistent and proclaims C the winner. We allow ties for the second and third place.

(b) Do the same but without allowing ties for the second and third place.

Hint: Reduce the problem to Integer Linear Programming problem and then use a software package like *Mathematica* to solve it. Mathematica is free for students and available on the UNSW IT webpage. If you have never used it before and try it out now, you are guaranteed that you will not be able to live without it! Your minimisation problem in Mathematica would look something like this:

$$\text{Minimize}[\{a+b+c+d+e+f, \text{some inequalities here}, \{a,b,c,d,e,f\} \in \text{Integers}\}, \{a,b,c,d,e,f\}]$$

So then, what voting procedure is the best? We design algorithms by first providing specifications which the input and the output have to meet. So let us try to figure out what conditions (or axioms) our voting method should satisfy.

1. Each personal preference list is complete and consistent in the sense that if $A > B$ and $B > C$ then $A > C$ (this property is usually called transitivity of an ordering relation).
2. The final output list is also complete, consistent and obtained deterministically, in the sense that the same input lists always produce the same output list.
3. If all voters except at most one prefer A to B then in the output ranking A should be above B.

4. Let A and B be two candidates; assume that we change some of the input preference lists, but in such a way that the ordering between A and B does not change on any of the input lists; then the ordering between the candidates A and B on the output list should not change either, regardless of how we change the ordering of other candidates.

The above axioms seem quite reasonable: 1 and 2 are basic consistency requirements; 3 enforces respect for a consensus as well as prohibits that a single voter can dictate the outcome of the election regardless of what all other voters wish; the last requirement might also appear natural: whether A should be placed above B should depend only on preferences of voters with respect to these two candidates; the preferences regarding other candidates should have no impact on the outcome of the order between A and B.

So perhaps we should be looking for a vote aggregation method which meets these axioms. Remarkably, it turns out that there is no vote aggregation method which meets all of the above axioms![1]

Theorem 0.1 (Arrow's Impossibility Theorem) *Assume that there are at least three candidates and at least two voters; then there is no vote aggregation method which satisfies all of the axioms 1-4.*

To obtain a simpler proof of (a version of) the Arrow's Theorem, we add another axiom which seems completely uncontroversial:

5 All candidates are treated equally, in the sense that if we relabel the candidates thus producing a permutation of the candidates, then the output list produced by the method will be permuted by the same permutation.

Theorem 0.2 (Terrence Tao) *Assuming that there are at least three candidates and at least two voters, there is no vote aggregation method which meets all of the axioms 1 - 5.*

Proof: Assume that there are N voters; let us define the notion of a *quorum* as any subset X of the set of all voters such that whenever all the voters in X put a candidate A above a candidate B then A must also appear above B in the voting outcome list. Clearly, by Axiom 3 every set of $N-1$ voters is a quorum.

We now show that for every two subsets of voters X and Y, if both X and Y are quorums then so is their intersection $X \cap Y$. Let A and B be any two candidates. Assume now that every voter in $X \cap Y$ prefers A to B. We have to show that in this case the output list A must also be above B. Assume that this is not the case; thus for a collection of voting lists L assume that the algorithm puts B above A. Let C be an arbitrary third candidate. We can now alter lists in L so that we do not change the order of candidates A and B, but we make sure that in all lists of voters in X candidate A is above candidate C, and that in all lists of voters from Y candidate C is above candidate B. By axiom 4 the final output list will not change and B will be still above A. However, since X is a quorum A must be put above C on the output list and since Y is a quorum C must be put above B, which would imply that A must be put above B, which is a contradiction.

We now consider $N-1$ quorums consisting of sets of $N-1$ voters which all contain voter V_1 and are missing precisely one of the voters $V_2, V_3, \ldots, V_N$. Their intersection must also be a quorum by what we have just proved. However, their intersection is the singleton V_1. This would imply that voter V_1 can determine the outcome of the election for any pair of candidates regardless of how other voters vote, which contradicts Axiom 3. □

Homework Problem 6.

1. We now weaken axiom 4 by strengthening its assumption as follows:

   4* Let A and B be two candidates; assume that we change some of the input preference lists, but in such a way that the ordering between A and B **as well as the number of candidates between A and B** does not change on any of the input lists; then the ordering between the candidates A and B on the output list should not change either, regardless of how we change the ordering of other candidates.

   Show that the above proof no longer works. Ray Li and Ishraq Huda, who took COMP4121 in 2016, have proved the following two theorems:

2. Prove that if the number of candidates is larger or equal to the number of voters, then there is no voting aggregation scheme which satisfies thus modified axioms.

3. Prove that if the number of voters is larger than the number of candidates then the Borda Count satisfies all such modified axioms.

[1] There are several versions of the Arrow's Theorem, including by Arrow himself.

1 Adaptive voting schemes

Rather that imposing axiomatic conditions on what voting schemata must satisfy, we can look for voting schemata which appear reasonable but have an extra robustness property against voting manipulations. Thus, assume that a subset of voters actually do not sincerely care about all available candidates but only wish to ensure that the candidate of their choice wins, and simply put their preferred candidate first and all other candidates in a random order; we would like to discount such votes as manipulative. In a sense, the goal of such a voting scheme is a final voting outcome which is as objective representation of a community sentiment as possible.

Such a voting schema would be useful for purposes other than voting for the best candidate; for example for customer feedback used to rank products in e-commerce, for ranking movies, etc.

There are two setups where the method which we are going to propose applies. In the first setup, each voter can give a rating to (some of the) candidates. The rating scale can be 0 to $M-1$ or 1 to M, where M is the number of candidates. Clearly, if every voter has to give a different rating to each candidate and if each voter is supposed to rank all of the candidates, then this is equivalent to a voter providing an ordered list of candidates, say in a decreasing order of preference, just like we had when we considered the three types of vote aggregation algorithms. However, this is not necessarily required; each voter can be free to rank only some of the candidates, also giving the same rank to several of them; moreover, the range of rankings can be much smaller than the number of candidates. One example of such voting is movie ranking, where each user "votes" for the number of stars to be given to a particular movie, usually between zero (or one) and up to 5 (or 10). So in this case each voter can be seen as casting his vote for what he thinks is a fair number of stars a movie should get. The goal of the voting aggregation procedure would be to produce credible ranks of all candidates, with ties allowed, which is robust with respect to collusion attacks, where several voters collude with a strategy to influence the final scores.

In the second setup the choices need not be linearly ordered in any natural way. For example, we could have an election for several positions in a city council. For each position we can have a list with several candidates, and each voter can chose one candidate on each list for every position to be filled.[1]

We now make the details more precise.

Setup: We will assume that a set of N *voters* $V_1, \ldots, V_N$ are given a collection of L lists $\Lambda_1, \ldots, \Lambda_L$; each list Λ_l contains n_l many candidates $\Lambda_l = \{I^l_1, \ldots, I^l_{n_l}\}$, and the voters are asked to chose the "best" candidate on each list. Not every voter is obliged to vote for the best candidate on every list, but can choose to vote on a subset of these lists.

Problem: As the system receives these votes, the task is to assess:

1. the level of "community approval" for each candidate on every list.
2. the reliability or *trustworthiness* of each voter, i.e., the degree to which his choices agree with the sentiment of the community.

In order to make such estimate of the level of "community approval" robust against possible unfair voting practices of the participants, the assessment of the trustworthiness of voters should have the following features (at the moment we allow these features to be specified both very vaguely and in a somewhat circular way).

1. Voters who vote on a large number of lists, and whose choices are largely in agreement with the prevailing sentiment of the community of voters, should obtain a higher level of trustworthiness than the voters who vote on only a few of these lists, or vote incompatibly with the prevailing sentiment; moreover, impact on the final outcome of each voter should be proportional to (or positively correlated with) his trustworthiness.
2. Voters who seldom vote or voters who vote more or less randomly, just to be seen as active in the community, and then choose to vote unfairly on a few particular lists for candidates which are favoured by few other voters, should not be able to secure election of their choices even if such colluding voters are a majority for those particular lists.

One might feel that such vote aggregation is not very democratic and that all voters should have equal impact regardless of how similar to a prevailing sentiment their vote is. In fact, our voting was not really designed for genuine social choices, but for aggregation of customer opinion. There has been a research that points out that fraudulent "voting" in the form of evaluating products or sellers on websites such as eBay is rampant and that it often renders customer feedback useless. Our vote aggregation algorithm was designed with an aim to mitigate precisely such problems.

[1] Even this can be further generalised by allowing voters to split their votes, giving fractional weight to several candidate on the same list.

Algorithm 1 Adaptive Voting Algorithm

Initialization: Let $\varepsilon > 0$ be the precision threshold, $\alpha \geq 1$ a discrimination setting parameter and

$$\begin{aligned} T_i^{(0)} &= 1; \\ \rho_{li}^{(0)} &= \frac{\sum_{r\,:\,r\to li} T_r^{(0)}}{\sqrt{\sum_{1\leq j\leq n_l}\left(\sum_{r\,:\,r\to lj} T_r^{(0)}\right)^2}} \\ &= \frac{|\{r\,:\,r\to li\}|}{\sqrt{\sum_{1\leq j\leq n_l}|\{r\,:\,r\to lj\}|^2}}. \end{aligned}$$

Repeat:

$$T_r^{(k+1)} = \sum_{l,i\,:\,r\to li} \rho_{li}^{(k)}; \tag{1.1}$$

$$\rho_{li}^{(k+1)} = \frac{\sum_{r\,:\,r\to li}\left(T_r^{(k+1)}\right)^\alpha}{\sqrt{\sum_{1\leq j\leq n_l}\left(\sum_{r\,:\,r\to lj}\left(T_r^{(k+1)}\right)^\alpha\right)^2}}; \tag{1.2}$$

until: $\|\boldsymbol{\rho}^{(k+1)} - \boldsymbol{\rho}^{(k)}\|_2 < \varepsilon$.

1.1 An Iterative Vote Aggregation Algorithm

Let us first introduce some notation:

- $r \to li$ denotes the fact that voter V_r has participated in voting for the best candidate on list Λ_l and has chosen candidate I_i^l;
- n_l denotes the number of candidates on a list l; thus, $n_l = |\Lambda_l|$;
- for each candidate I_i^l on list Λ_l we will keep track of its *level of community approval* at the step of iteration k, denoted by $\rho_{li}^{(k)}$;
- these individual community approval levels $\rho_{li}^{(k)}$ will be collected into a single vector

  $$\boldsymbol{\rho}^{(k)} = \langle \rho_{li}^{(k)} \;:\; 1 \leq l \leq L,\ 1 \leq i \leq n_l \rangle;$$

  thus, if we let $M = \sum_{1\leq l\leq L} n_l$, then $\boldsymbol{\rho}^{(k)} \in \mathbb{R}^M$;
- for any vector $\boldsymbol{\rho} \in \mathbb{R}^M$ we define $(\boldsymbol{\rho})_l$ to be the projection $\langle \rho_{li} \;:\; 1 \leq i \leq n_l \rangle$ of $\boldsymbol{\rho}$ to the subspace corresponding to a single list Λ_l.
- for each voter V_r we will also keep track of his *trustworthiness* $T_r^{(k)}$ at the stage of iteration p.
- for each $p \geq 1$ we denote by $\|\boldsymbol{x}\|_p$ the usual p-norm of the vector $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$, i.e.,

  $$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{\frac{1}{p}}.$$

Algorithm 1 shows our iterative vote aggregation algorithm; k denotes the stage of iteration. We now explain the intuitive motivation behind it. We start with the usual vote count, where every voter has one vote, equal in value to votes of all other voters. For each candidate $I_i^l \in \Lambda_l$ on a list Λ_l, the initial rank $\rho_{li}^{(0)}$ of I_i^l is simply the number of votes which this candidate has received, normalized so that $\sum_j \rho_{lj}^2 = 1$. In the next round of iterative aggregation of the votes, each voter first gets its trustworthiness rank, which is the sum total of the ranks of all the candidates which he has voted for, "prorated" by a monotonically increasing function $f(x) = x^\alpha$; we will later explain such choice of $f(x)$.

The idea is that now voters themselves can be judged by the selections they have made; a high trustworthiness rank will be given only to voters who have often chosen candidates favoured by many other members of the community, thus voting in accordance with the prevailing community sentiment. Such voters can be considered as "reliable voters", choosing candidates in accordance with community sentiment. On the other hand, those who incorrectly judge others (meaning: not in accordance with the prevailing community sentiment) will themselves receive a low trustworthiness rank.[2]

Now we recalculate the ranks of candidates using (1.2); thus each received vote is now worth the present trustworthiness rank of the voter giving such vote. We continue such iterations until the ranks stop changing significantly, i.e., we stop when $\|\boldsymbol{\rho}^{(k+1)} - \boldsymbol{\rho}^{(k)}\|_2 < \varepsilon$, where ε is a threshold corresponding to a desired precision adequate for the particular application; in our experiments ε was in the range $10^{-4} - 10^{-8}$ with the algorithm terminating after $8 - 20$ iterations.

The value of the parameter α determines the robustness of our algorithm against unfair voters. Clearly, higher values of α increasingly favour voters with a high compliance with the prevailing community sentiment and penalise harsher for votes given to less favoured candidates. While this makes our system more robust, large values of α increasingly marginalise a significant fraction of honest, but less successful voters. In our experiments the values $\alpha < 1.5$ were insufficient to obtain satisfactory robustness; the values $1.5 < \alpha < 3$ gave excellent performance without marginalising a large number of voters.

Our algorithm will eventually terminate with any strictly increasing, twice continuously differentiable function $f(x)$ in place of x^α, but we saw little value in such more general choices. As it will be obvious from the convergence proof below, the choice $f(x) = x^\alpha$ has a natural motivation, defining a commonly used norm on the vector space of trustworthiness ranks.

You can find at the class website a Mathematica implementation of the above algorithm and see how it performs in a case when a large group of malicious voters collude to promote their favourite candidate.

We now rigorously prove that our algorithm converges and characterise the values $\boldsymbol{\rho}$ which it produces.

1.2 Convergence proof

For a given *community approval* ranking $\boldsymbol{\rho}$ of candidates, referred in the sequel simply as *ranks of candidates*, let us define the corresponding *trustworthiness* $T_r(\boldsymbol{\rho})$ of a voter V_r as

$$T_r(\boldsymbol{\rho}) = \sum_{p,k\,:\,r\to pk} \rho_{pk}, \tag{1.3}$$

and denote by $\boldsymbol{T}(\boldsymbol{\rho})$ the vector of such trustworthiness ranks, $\boldsymbol{T}(\boldsymbol{\rho}) = \langle T_1(\boldsymbol{\rho}), T_2(\boldsymbol{\rho}), \ldots, T_N(\boldsymbol{\rho})\rangle$; recall that[3]

$$\|\boldsymbol{T}\|_{\alpha+1} = \left(\textstyle\sum_r T_r^{\alpha+1}(\boldsymbol{\rho})\right)^{\frac{1}{\alpha+1}}$$

is the $\alpha+1$-norm of the vector $\boldsymbol{T}(\boldsymbol{\rho}) \in \mathbb{R}^N$.

For a reason which will be clear later, we now assign the ranks to the candidates ranked, in such a way that:

1. for every list Λ_l, the vector of ranks of all candidates on that list is a unit vector, i.e., $\|(\boldsymbol{\rho})_l\|_2 = 1$;
2. the $\alpha+1$ norm $\|\boldsymbol{T}\|_{\alpha+1}$ of the trustworthiness vector $T(\boldsymbol{\rho})$ is maximized.

The reason for considering $\|\boldsymbol{T}(\boldsymbol{\rho})\|_{\alpha+1}$ rather than $\|\boldsymbol{T}(\boldsymbol{\rho})\|_\alpha$ in the second condition will be clear below. Note that, in a sense, this gives "the benefit of the doubt" to all voters, giving them the largest possible joint trustworthiness rank (in terms of the $\alpha+1$ norm of the trustworthiness vector) which is consistent with their votes. On the other hand, since we are not estimating the "absolute quality" of candidates, but only their relative ordering according to their perceived quality, we maintain for each list the same, unit 2-norm of the vector corresponding to ranks of all candidates on that list; this is done to ensure convergence of our algorithm. Thus, if we now define

$$F(\boldsymbol{\rho}) = (\|\boldsymbol{T}(\boldsymbol{\rho})\|_{\alpha+1})^{\alpha+1} = \sum_{r=1}^{N} \left(\sum_{p,k\,:\,r\to pk} \rho_{pk} \right)^{\alpha+1} \tag{1.4}$$

then our aim is equivalent to maximizing $F(\boldsymbol{\rho})$, subject to the constraints

$$\mathcal{C} = \left\{ \sum_{i=1}^{n_l} \rho_{li}^2 = 1, \quad 1 \le l \le L \right\},$$

[2] We cannot help mentioning that this idea is remarkably old: "Judge not, and you shall not be judged ..." as well as "Do not judge, or you too will be judged. For in the same way you judge others, you will be judged, and with the measure you use, it will be measured to you." (The New Testament, Luke 6:37 and Matthew 7:1, respectively).

[3] Note that $\boldsymbol{T} > 0$, so no need for absolute values in the sum for the α-norm of $\boldsymbol{T}$.

For this purpose we introduce for each list Λ_l a Lagrangian multiplier λ_l, define $\boldsymbol{\lambda} = \langle \lambda_l, : 1 \le l \le L \rangle$ and look for the stationary points of the Lagrangian function[4]

$$\Phi(\boldsymbol{\rho}, \boldsymbol{\lambda}) = F(\boldsymbol{\rho}) - \sum_{q=1}^{L} \lambda_q \left(-1 + \sum_{m=1}^{n_q} \rho_{qm}^2 \right).$$

Let l, m be list indices and i, j candidate indices; then, using the fact that the trustworthiness functions $T_r(\boldsymbol{\rho})$ are linear, we obtain

$$\frac{\partial F(\boldsymbol{\rho})}{\partial \rho_{li}} = (\alpha + 1) \sum_{r \,:\, r \to li} T_r^{\alpha}(\boldsymbol{\rho}); \tag{1.5}$$

$$\frac{\partial \Phi(\boldsymbol{\rho}, \boldsymbol{\lambda})}{\partial \rho_{li}} = (\alpha + 1) \sum_{r \,:\, r \to li} T_r^{\alpha}(\boldsymbol{\rho}) - 2\lambda_l \rho_{li} \tag{1.6}$$

$$\frac{\partial^2 F(\boldsymbol{\rho})}{\partial \rho_{li} \partial \rho_{mj}} = \alpha(\alpha + 1) \sum_{r \,:\, r \to li, mj} T_r^{\alpha - 1}(\boldsymbol{\rho}). \tag{1.7}$$

Note that, since every voter votes for at most one candidate on each list, $\frac{\partial^2 F(\boldsymbol{\rho})}{\partial \rho_{li} \partial \rho_{lj}} = 0$ for every l, i, j such that $i \neq j$. Also, if $(\boldsymbol{\rho}, \boldsymbol{\lambda})$ is a stationary point of Φ then $\frac{\partial \Phi(\boldsymbol{\rho}, \boldsymbol{\lambda})}{\partial \rho_{li}} = 0$ and thus from (1.6) we get

$$\rho_{li} \lambda_l = \frac{\alpha + 1}{2} \sum_{r \,:\, r \to li} T_r^{\alpha}(\boldsymbol{\rho}). \tag{1.8}$$

This yields

$$\rho_{li}^2 \lambda_l^2 = \frac{(\alpha + 1)^2}{4} \left(\sum_{r \,:\, r \to li} T_r^{\alpha}(\boldsymbol{\rho}) \right)^2,$$

and by summing the above equations for all indices i of candidates on the list l we get

$$\lambda_l{}^2 \sum_{i=1}^{n_l} \rho_{li}^2 = \frac{(\alpha + 1)^2}{4} \sum_{m=1}^{n_l} \left(\sum_{r \,:\, r \to lm} T_r^{\alpha}(\boldsymbol{\rho}) \right)^2.$$

Since $(\boldsymbol{\rho}, \boldsymbol{\lambda})$ is a stationary point of Φ also $\frac{\partial \Phi(\boldsymbol{\rho}, \boldsymbol{\lambda})}{\partial \lambda_l} = 0$; this implies $\sum_{i=1}^{n_l} \rho_{li}^2 = 1$. Thus, the above equation becomes

$$\lambda_l{}^2 = \frac{(\alpha + 1)^2}{4} \sum_{m=1}^{n_l} \left(\sum_{r \,:\, r \to lm} T_r^{\alpha}(\boldsymbol{\rho}) \right)^2.$$

Since by (1.8) λ_l must be positive, this yields

$$\lambda_l = \frac{\alpha + 1}{2} \sqrt{\sum_{m=1}^{n_l} \left(\sum_{r \,:\, r \to lm} T_r^{\alpha}(\boldsymbol{\rho}) \right)^2}.$$

which, together with (1.8) yields

$$\rho_{li} = \frac{\sum_{r \,:\, r \to li} T_r^{\alpha}(\boldsymbol{\rho})}{\sqrt{\sum_{m=1}^{n_l} \left(\sum_{r \,:\, r \to lm} T_r^{\alpha}(\boldsymbol{\rho}) \right)^2}}. \tag{1.9}$$

We now define $\boldsymbol{\rho} \mapsto \boldsymbol{\rho}^*$ to be the mapping such that for an arbitrary vector $\boldsymbol{\rho}$, a list Λ_l and item i,

$$(\boldsymbol{\rho}^*)_{li} = \frac{\sum_{r \,:\, r \to li} T_r^{\alpha}(\boldsymbol{\rho})}{\sqrt{\sum_m \left(\sum_{r \,:\, r \to lm} T_r^{\alpha - 1}(\boldsymbol{\rho}) \right)^2}}. \tag{1.10}$$

Consequently, $(\tilde{\boldsymbol{\rho}}, \tilde{\boldsymbol{\lambda}})$ is a stationary point of Φ just in case $\tilde{\boldsymbol{\rho}}^* = \tilde{\boldsymbol{\rho}}$, i.e., for all $1 \le l \le L$,

$$(\tilde{\boldsymbol{\rho}})_{li} = \frac{\sum_{r \,:\, r \to li} T_r^{\alpha}(\tilde{\boldsymbol{\rho}})}{\sqrt{\sum_m \left(\sum_{r \,:\, r \to lm} T_r^{\alpha - 1}(\tilde{\boldsymbol{\rho}}) \right)^2}}. \tag{1.11}$$

[4] If you are not familiar with constrained optimisation using Lagrangian multiplies you might want to read about it in an article in Wikipedia: `https://en.wikipedia.org/wiki/Lagrange_multiplier`.

(a) case when h is sufficiently small

(b) additional initial line search

Figure 1.1: A geometric representation of the iterative procedure

Thus, our solution for maximising the $\alpha + 1$ norm of the trust vector $\mathbf{T}(\rho)$ is a fixed point for the mapping we use in our iterative process! We now want to prove that an iterative procedure with such a mapping indeed converges and thus eventually results in a good approximation of a fixed point.

Recall that we denote by $(\boldsymbol{x})_l$ the projection of a vector $\boldsymbol{x} \in \mathbb{R}^M$ to the subspace of dimension n_l, which corresponds to a list Λ_l. Thus, using (1.5), equations (1.10) can be written as[5]

$$(\boldsymbol{\rho}^*)_l = \frac{(\nabla F(\boldsymbol{\rho}))_l}{\|(\nabla F(\boldsymbol{\rho}))_l\|_2}. \tag{1.12}$$

Note that this equation has a nice geometric interpretation of the mapping $\boldsymbol{\rho} \mapsto \boldsymbol{\rho^*}$: given any vector $\boldsymbol{\rho}$ the new vector $\boldsymbol{\rho^*}$ is a vector which has the property that its orthogonal projection to the subspace which correspond to the list Λ_l, i.e., $(\boldsymbol{\rho^*})_l$, is the unit vector parallel to the gradient $\nabla F(\boldsymbol{\rho})$ of $F(\boldsymbol{\rho})$ at point $\boldsymbol{\rho}$; see Figure 1.1(a).

So we found out that $(\tilde{\boldsymbol{\rho}}, \tilde{\boldsymbol{\lambda}})$ is a stationary point of Φ just in case $\tilde{\boldsymbol{\rho}}^* = \tilde{\boldsymbol{\rho}}$, i.e., for all $1 \le l \le L$,

$$(\tilde{\boldsymbol{\rho}})_l = \frac{(\nabla F(\tilde{\boldsymbol{\rho}}))_l}{\|(\nabla F(\tilde{\boldsymbol{\rho}}))_l\|_2}. \tag{1.13}$$

Thus, the desired solution is a fixed point of the mapping $\boldsymbol{\rho} \mapsto \boldsymbol{\rho}^*$, but unlike the PageRank, in this case, due to the presence of the denominator which also involves $\boldsymbol{\rho}$, the mapping, given by (1.10), is non linear in the variables $\boldsymbol{\rho}$.

Note also that in our algorithm the approximation $\boldsymbol{\rho}^{(n+1)}$ of the vector $\boldsymbol{\rho}$ obtained at the stage of iteration $(n+1)$ can be written as

$$\boldsymbol{\rho}^{(n+1)} = (\boldsymbol{\rho}^{(n)})^*,$$

and that our algorithm will halt when $\boldsymbol{\rho}^{(n)}$ get sufficiently close to a fixed point $\tilde{\boldsymbol{\rho}} = \tilde{\boldsymbol{\rho}}^*$ of the mapping $\boldsymbol{\rho} \to \boldsymbol{\rho}^*$.

Let $\boldsymbol{\rho}$ be an arbitrary vector such that $\|(\boldsymbol{\rho})_l\|_2 = 1$ for all $1 \le l \le L$; we let $\boldsymbol{h} = \boldsymbol{\rho}^* - \boldsymbol{\rho}$. By applying the Taylor formula with the remainder in the Lagrange form, we get that for some $0 < c < 1$ and $\boldsymbol{\mu}_c = c\boldsymbol{\rho} + (1-c)\boldsymbol{\rho}^*$ We have[6]

$$\begin{aligned} F(\boldsymbol{\rho}^*) &= F(\boldsymbol{\rho} + \boldsymbol{h}) \\ &= F(\boldsymbol{\rho}) + \nabla F(\boldsymbol{\rho}) \cdot \boldsymbol{h} + \frac{1}{2} \sum_{l,m,i,j} \frac{\partial^2 F(\boldsymbol{\mu}_c)}{\partial \rho_{li} \partial \rho_{mj}} h_{li} h_{mj}. \end{aligned} \tag{1.14}$$

Since

$$(\boldsymbol{h})_l = (\boldsymbol{\rho}^*)_l - (\boldsymbol{\rho})_l = \frac{(\nabla F(\boldsymbol{\rho}))_l}{\|(\nabla F(\boldsymbol{\rho}))_l\|_2} - (\boldsymbol{\rho})_l, \tag{1.15}$$

[5] Note that $\nabla F(\boldsymbol{\rho})$ denotes the gradient of $F(\boldsymbol{\rho})$, i.e., a vector whose (l,i) coordinate is the partial derivative $\partial F(\boldsymbol{\rho})/\partial \rho_{li}$.

[6] Note that here the dot represents the scalar product.

using also (1.12), we get

$$\begin{aligned}(\nabla F(\boldsymbol{\rho}))_l \cdot (\boldsymbol{h})_l &= (\nabla F(\boldsymbol{\rho}))_l \cdot \left(\frac{(\nabla F(\boldsymbol{\rho}))_l}{\|(\nabla F(\boldsymbol{\rho}))_l\|_2} - (\boldsymbol{\rho})_l \right) \\ &= \frac{\|(\nabla F(\boldsymbol{\rho}))_l\|_2^2}{\|(\nabla F(\boldsymbol{\rho}))_l\|_2} - (\nabla F(\boldsymbol{\rho}))_l \cdot (\boldsymbol{\rho})_l \\ &= \|(\nabla F(\boldsymbol{\rho}))_l\|_2 - (\nabla F(\boldsymbol{\rho}))_l \cdot (\boldsymbol{\rho})_l \\ &= \|(\nabla F(\boldsymbol{\rho}))_l\|_2 - \|(\nabla F(\boldsymbol{\rho}))_l\|_2 (\boldsymbol{\rho}^*)_l \cdot (\boldsymbol{\rho})_l \\ &= \|(\nabla F(\boldsymbol{\rho}))_l\|_2 (1 - (\boldsymbol{\rho}^*)_l \cdot (\boldsymbol{\rho})_l) \end{aligned} \tag{1.16}$$

Let θ_l be the angle between the unit vectors $(\boldsymbol{\rho})_l$ and $(\boldsymbol{\rho}^*)_l$, i.e., such that $\cos\theta_l = (\boldsymbol{\rho})_l \cdot (\boldsymbol{\rho^*})_l$. Then, (see Figure 1.1(a))

$$\left\| \frac{(\boldsymbol{h})_l}{2} \right\|_2^2 = \left(\sin \frac{\theta_l}{2} \right)^2 = \frac{1 - \cos\theta_l}{2} = \frac{1 - (\boldsymbol{\rho})_l \cdot (\boldsymbol{\rho}^*)_l}{2}. \tag{1.17}$$

Combining (1.16) and (1.17) we get

$$(\nabla F(\boldsymbol{\rho}))_l \cdot (\boldsymbol{h})_l = \frac{\|(\nabla F(\boldsymbol{\rho}))_l\|_2 \, \|(\boldsymbol{h})_l\|_2^2}{2}. \tag{1.18}$$

Assume first that $\|\boldsymbol{h}\|_2$ is sufficiently small, so that the contribution of the second order terms in (1.14) is small compared to the first order term, and, consequently

$$F(\boldsymbol{\rho}^*) \approx F(\boldsymbol{\rho}) + \nabla F(\boldsymbol{\rho}) \cdot \boldsymbol{h}. \tag{1.19}$$

Since $\|\nabla F(\boldsymbol{\rho})\|_2$ is a continuous function, it achieves its minimum on the compact set defined by our constraints, i.e., on the set $\mathcal{C} = \{\boldsymbol{\rho} \,:\, \|(\boldsymbol{\rho})_l\| = 1,\ 1 \le l \le L\}$. It is easy to see that the directional derivative of $F(\boldsymbol{\rho})$ in the (radial) direction of vector $\boldsymbol{\rho}$ is always strictly positive; thus, $\|\nabla F(\boldsymbol{\rho})\|_2$ is also positive and consequently, on the compact set $\mathcal{C}$ the minimum of $\|\nabla F(\boldsymbol{\rho})\|_2$ must also be strictly positive. Thus, there exists $\kappa > 0$ such that $\|\nabla F(\boldsymbol{\rho})\|_2 > \kappa$ for all $\boldsymbol{\rho} \in \mathcal{C}$; using this and by summing equations (1.18) for all $1 \le l \le L$, we get

$$\nabla F(\boldsymbol{\rho}) \cdot \boldsymbol{h} > \frac{\kappa \, \|\boldsymbol{h}\|_2^2}{2}. \tag{1.20}$$

This and (1.19) imply that, for $\boldsymbol{\rho}^{(n)}$ and $\boldsymbol{h}^{(n)} = \boldsymbol{\rho}^{(n+1)} - \boldsymbol{\rho}^{(n)}$ obtained in our iterations,

$$F(\boldsymbol{\rho}^{(n+1)}) - F(\boldsymbol{\rho}^{(n)}) = F((\boldsymbol{\rho}^{(n)})^*) - F(\boldsymbol{\rho}^{(n)}) > \frac{\kappa \, \|\boldsymbol{h}^{(n)}\|_2^2}{2}. \tag{1.21}$$

By summing such equations for all $k = 0..n$ we get that

$$\begin{aligned} F(\boldsymbol{\rho}^{(n)}) - F(\boldsymbol{\rho}^{(0)}) &= (F(\boldsymbol{\rho}^{(n)}) - F(\boldsymbol{\rho}^{(n-1)})) + (F(\boldsymbol{\rho}^{(n-1)}) - F(\boldsymbol{\rho}^{(n-2)})) + \ldots + (F(\boldsymbol{\rho}^{(1)}) - F(\boldsymbol{\rho}^{(0)})) \\ &> \frac{\kappa \, \sum_{k=0}^{n-1} \|\boldsymbol{h}^{(k)}\|_2^2}{2}. \end{aligned} \tag{1.22}$$

Since $F(\boldsymbol{\rho})$ is continuous on a compact set $\mathcal{C}$ defined by the constraints, it must also be bounded on $\mathcal{C}$. Consequently, the sum on the right of (1.22) must be convergent, and this implies not only that $\|\boldsymbol{h}^{(k)}\|_2^2$ goes to zero as $n \to \infty$, but also that it converges fast (to make the sum convergent as well). Thus, since $\boldsymbol{h}^{(n)} = \boldsymbol{\rho}^{(n+1)} - \boldsymbol{\rho}^{(n)}$, we see that eventually $\|\boldsymbol{\rho}^{(n+1)} - \boldsymbol{\rho}^{(n)}\|$ will be smaller than the prescribed threshold and the algorithm will terminate.

If $\|\boldsymbol{h}\|_2$ is not sufficiently small so that the impact of the second order term in (1.14) makes the inequality (1.20) false, we supplement our algorithm with an initial phase which involves a line search. While this ensures provable convergence of our algorithm, in all of our (very numerous) experiments such a line search was never activated; however, we were unable to prove without any additional assumptions that indeed such line search is superfluous, so we present a slight modification of our algorithm. Let

$$f(\boldsymbol{\rho}, t) = F(\boldsymbol{\rho} + t(\boldsymbol{\rho}^* - \boldsymbol{\rho}));$$

then, by the previous considerations, for sufficiently small t function $f(\boldsymbol{\rho}^{(n)}, t)$ is increasing in t. We now modify our iteration step as follows. If there exists $t_0 \in (0, 1)$ such that $\frac{\partial f}{\partial t}(\boldsymbol{\rho}, t_0) = 0$ (testing this amounts to solving

a low degree algebraic equation), then we let $\boldsymbol{\rho}^{(n+1)} = \boldsymbol{\rho}'$, where $\boldsymbol{\rho}'$ is defined so that for all $1 \leq l \leq L$, and for the smallest root t_0 of the above equation,

$$(\boldsymbol{\rho}')_l = \frac{(\boldsymbol{\rho}^{(n)})_l + t_0((\boldsymbol{\rho}^{(n)*})_l - (\boldsymbol{\rho}^{(n)})_l)}{\|(\boldsymbol{\rho}^{(n)})_l + t_0((\boldsymbol{\rho}^{(n)*})_l - (\boldsymbol{\rho}^{(n)})_l)\|_2};$$

see Figure 1.1(b); if no such t_0 exists, we let

$$\boldsymbol{\rho}^{(n+1)} = (\boldsymbol{\rho}^{(n)})^*.$$

The convergence now follows from an argument similar to the one in the previous case.

1.3 Rating Through Voting Procedure

We now describe how we use the proposed voting algorithm in online rating systems, evaluating some sort of products..

Our rating procedure starts by assigning to each product π_l, $(1 \leq l \leq L)$, a voting list Λ_l; the candidates on each voting list are the rating levels, comprising a "scale" from 1 to n, (in practice $n \leq 10$). Each rater is now construed as a voter V_r, $(1 \leq r \leq N)$. We then process the evaluations, by interpreting an evaluation of level i, $(1 \leq i \leq n)$, of a product π_l, $(1 \leq l \leq L)$, by a voter V_r, as his vote for candidate i on the list Λ_l (recall that the candidates on each election list are the corresponding evaluation levels). After our iterative algorithm has terminated, each level $1 \leq i \leq n$ on each voting list Λ_l has received a corresponding credibility degree ρ_{li}.

We can now obtain a rating score $\mathrm{R}(\pi_l)$ for each product π_l, using such credibility degrees $(\boldsymbol{\rho})_l = \langle \rho_1, \ldots, \rho_n \rangle$ in a way which suits the particular application best. For example, if the rating scores have to reflect where the community sentiment is centered, we can simply choose as the rating score $\mathrm{R}(\pi_l)$ of π_l the rating level which has the highest credibility rank. Such a rating score does not involve any averaging and is most indicative of the community's *prevailing sentiment*. On the other hand, if we wish to obtain a score which emphasises such prevailing sentiment, but, to a varying degree, taking into account "dissenting views", one can form a weighted average of the form

$$\mathrm{R}(\pi_l) = \sum_{1 \leq i \leq n_l} \frac{\rho_{li} \ \times i}{\sum_{1 \leq j \leq n_l} \rho_{lj}}.$$

Yet another possibility, which appears to work the best in practice is as follows. Let $v(r, l)$ be equal to i such that $r \to l, i$ (i.e., $v(r, l)$ is the level chosen on list l by voter V_r). Then we can define

$$\mathrm{R}(\pi_l) = \sum_{r : r \to l} \frac{T_r \times v(r, l)}{\sum_{r : r \to l} T_r};$$

in this way the score given to a product π_l is a weighted average of scores given by all voters who have evaluated product π_l, prorated by the trustworthiness of the voter.

Please look carefully at Mathematica implementation of the above voting algorithm in the file included on the course webpage which illustrates how effectively the algorithm counters a massive collusion attack. Also in the paper on the class web site under Topic 2 you can se an evaluation of a movie rating system built by my former PhD student Mohammad Allahbakhsh.

The voting algorithm we presented can be improved in many ways, some of which are completely unexplored. So you might consider playing with the ideas presented for your major project.

Further Reading

- Mung Chiang: Networked Life, Cambridge University Press, 2012; please read Chapters 5 and 6 entitled *"When can I trust an average rating on Amazon?"* and *"Why Does Wikipedia even work?"*.
- Terrence Tao: Arrow's Theorem, https://www.math.ucla.edu/~tao/arrow.pdf
- M. Allahbakhsh and A. Ignjatovic: An Iterative Method for Calculating Robust Rating Scores, IEEE Transactions on Parallel and Distributed Systems, 26(2), (2015) pp. 340-350.