Visual Product Discovery

Diego Legrand, Philip M. Long, Myles Brundage,
Tom Angelopoulos, Olivier Francon, Vinit Garg, Wallace Mann, Vivek Ramamurthy,
Antoine Saliou, Bethany Simmons, Peter Skipper, Petr Tsatsin, Richard Vistnes, and
Nigel Duffy*

Sentient Technologies
One California, Suite 2300
San Francisco, CA 94111 USA

ABSTRACT

We describe a system, Sentient Aware, that allows a user to interactively navigate through a catalog by viewing and clicking on images of products. When a user clicks on a product, she receives a new set of products to browse that is enriched for products that are similar to the clicked product. This continues, allowing the user to define an increasingly refined set of products, solely by expressing preferences between images of products.

We describe the design of Sentient Aware, including its rationale, and some experiments. We also discuss limitations of our model of the problem, and potential alternatives.

CCS Concepts

•**Applied computing** → **Online shopping;**

Keywords

Information retrieval; on-line shopping

1. INTRODUCTION

While shopping online offers many advantages including convenience and price comparison, searching through large catalogs to find exactly what you want can be challenging. The typical keyword-based searching methods require consistent tagging and, even under the best of conditions, will still break down if the shopper does not have the same vocabulary as the retailer. This leaves the shopper scrolling through a long list of products trying to find something suitable. Filtering by attributes such as category, brand and price is a common approach to narrow down the user's search. However, it is often hard for users to express their desires through filters in a way that restricts the catalog sufficiently. Filtering also creates hard boundaries between products that can be very similar and it is prone to mislabeling.

This paper is about a system that provides an alternative way to navigate a catalog of products. Rather than describing what she wants with a query or filters, the user implicitly expresses preferences by simply clicking on products that she likes. As she clicks she provides the system with progressively refined information about what she wants, so that it can enrich its results accordingly. We describe a product that provides this functionality, the design of an implementation of the product and its rationale, and some experiments. To focus on the main ideas, we have omitted some details of the production system.

2. THE PRODUCT

Our product is called Sentient Aware. It has been in production, with our first partner `SHOEme.ca`, since November, 2015. The original launch was with a catalog of women's boots; it is now used for nine additional shoe categories. The most basic version of Sentient Aware works as follows (see Figure 1). A shopper who enters the system is presented with a screen of diverse product images and is invited to click on the product that is most similar to what she is looking for. After she clicks, she is given the option to see more information about the clicked shoe, or buy it. She also is presented with another screen that is enriched for products that are similar to the product that she clicked on, and she is invited to click again. If she clicks on an image in this second screen, she once again receives an information/buy invitation, and a third set of products is assembled that is now enriched for products similar to both of her clicks (and dissimilar to images that were not clicked). This continues, with the system offering the user screens of products with a progressively refined scope informed by the preferences expressed by the user.

Our partners interact with our service as follows. First, they provide us with a catalog, which includes an image for each product, along with metadata, such as size. After our system processes the catalog, it exposes an API. The client repeatedly sends the user action to our service, which responds with a new list of products to show to the user.

*This work was done while Bethany Simmons was at Sentient Technologies; her current address is Betabrand, 3435 Cesar Chavez St. #224, San Francisco, CA 94110. Email addresses: diego.legrand@sentient.ai, phil.long@sentient.ai, myles.brundage@sentient.ai, tom.angelopoulos@sentient.ai, olivier.francon@sentient.ai, vinit.garg@sentient.ai, wallace.mann@sentient.ai, vivek.ramamurthy@sentient.ai, antoine.saliou@sentient.ai, bethany@betabrand.com, peter.skipper@sentient.ai, petr.tsatsin@sentient.ai, richard.vistnes@sentient.ai, nigel.duffy@sentient.ai.

KDD Workshop on Fashion Meets Machine Learning August 13, 2016, San Francisco, CA, USA

DOI: 10.1145/1235

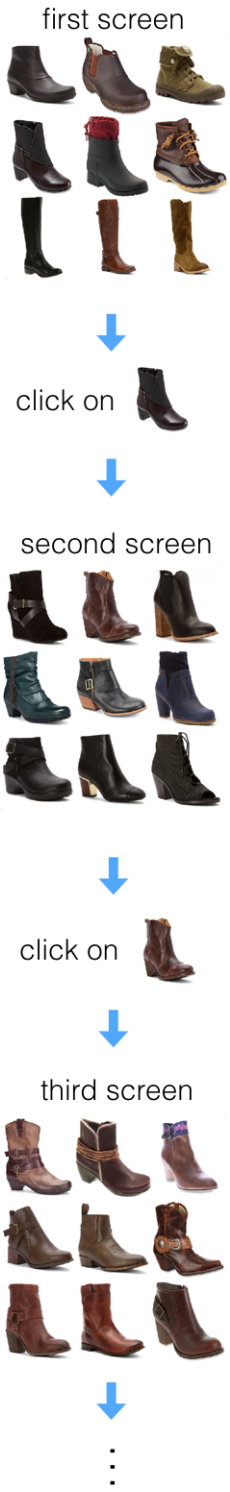

Figure 1: An example of a user session on Sentient Aware.

(Technically, to avoid the need to maintain state for multiple sessions, our API requires the client to recount all actions of the user at each iteration.)

We are working on a number of refinements of this experience that will be described in Section 10.

3. KEY MODULES

Our system is broken into modules that will be described in future sections.

Embedding training. Offline, before users interact with the system, it trains a model for scoring pairs of products for their dissimilarity. The output of this module is a mapping that takes as input a raw representation of the product such as an image, and outputs an embedding into $\mathbf{R}^{128}$. The goal of training is for the distance between the embeddings to reflect the dissimilarity between products.

Product scoring. At any given time, the system assigns a score to each product that reflects its estimated attractiveness to the user.

Screen selection. These attractiveness scores are used with the embeddings to choose a screen of products to present to the user. The goal here is to trade off appropriately between (a) presenting the user with products that are likely to be attractive, (b) providing the user with further opportunities to demonstrate preferences, and (c) reassuring the user that the system "understands" the preferences expressed with previous clicks.

4. USER MODEL

As in [4], our design is guided by the following model of a user. Before engaging with the system, the user has in mind a product in the catalog, which we will call the *target*. In each screen S, we model the probability that the user clicks on a product s given that the target is t using

$$\mathbf{Pr}(s|S,t) = \frac{\exp(-d(s,t))}{\sum_{s' \in S} \exp(-d(s',t))} \quad (1)$$

where $d(s,t)$ is the distance between the embeddings for s and t.

If the user is presented with a screen containing the target, the session ends. The goal is for this to happen as soon as possible.

5. LEARNING TO EMBED

Our offline training of embeddings builds on the work of Hoffer and Ailon [6]. They described a method for training a deep network to map images to points in $\mathbf{R}^d$, so that similar images are mapped to nearby points, and dissimilar images to far away points. Their network took as input triplets (t,u,v) of images, where a triplet was understood to mean that u is more similar to t than v is. The idea was for the deep network that used shared weights W to transform t, u and v to blocks $\phi_W(t), \phi_W(u)$ and $\phi_W(v)$ of k hidden nodes each, and then to update these weights using gradient descent to minimize a loss function that rewards making u closer to t than v is.

In a triplet, a judge has indicated which of two options is closer to t. In our application, a screen has more than two products. However, we may easily modify the algorithm of Hoffer and Ailon to train W to minimize $\log(1/P_W(s \mid S,t))$, where $P_W(s \mid S,t)$ is the probability, according to the model

of Section 4, that a judge regards s as the most similar element of S to t. This is equivalent to training the model of Section 4 using maximum likelihood, except that, of course, when using a rich hypothesis class like deep neural networks, we must regularize (e.g. add a term to the objective function that penalizes large weights – as in [6], we also use dropout [11]). Aside from training with screens instead of triplets, and changing the loss function to perform (penalized) maximum likelihood with respect to (1), we follow the design of Hoffer and Ailon.

5.1 Bootstrapping a rough model through uniform sampling

When building a model for a new catalog, to build a rough, preliminary model, we generate training data by picking a target t uniformly from the catalog, sampling S uniformly from subsets of a given size, and asking judges which member of S is most similar to t.

5.2 Refining the model through simulation

Typically, when we sample t and S as in Section 5.1, members of S are all quite dissimilar to t. With training data like this, the algorithm is not challenged to make the fine-grained distinctions that may be needed late in a user session.

We generate training data that balances between coarse- and fine-grained examples similarly to what will be required when the model is applied in the field as follows. We simulate user sessions using a preliminary model trained using data described in Section 5.1 along with a version of the entire system using that model: we pick a random target t, generate a screen S_1 using the system, then simulate the user click using the preliminary model, use the system to generate a new screen S_2, and continue. We then send (t, S_1), (t, S_2),... to judges. We repeat this for many targets t, and use the resulting data to train a new model. We have used the Mechanical Turk (`https://www.mturk.com`) for these judgments.

After we have trained a model with this new data, we could iterate this process again, using the new model to simulate the user.

6. SCORING

As in PicHunter [4], our scoring algorithm is based on Bayesian principles. It views the choice of the target as a random variable T. First, the system formulates a prior probability P_{prior} that each product is the target, then it updates this prior using the clicks of the user, based on the probability model described in Section 4. Specifically, if we use C to denote a random variable that reflects all of the user's clicks up to a given point in time, and c is an observation of C, we can use the model in Section 4 to calculate the probability $\mathbf{Pr}(C = c|T = t)$ that we would see the clicks that we did, given different possibilities t for the target. If we apply Bayes' rule together with the prior, the resulting posterior gives us exactly what we want to know:

$$\mathbf{Pr}(T = t|C = c) = \frac{\mathbf{Pr}(C = c|T = t) \times P_{\text{prior}}(T = t)}{\mathbf{Pr}(C = c)}.$$

By combining the user model with the assumption that the clicks on different screens are independent, we get

$$\mathbf{Pr}(C = c|T = t) = \prod_i \mathbf{Pr}(s_i|S_i, T = t)$$

where s_i is the product selected in the ith screen S_i.

Since the session ends when the target is included in any screen, at some point before the end of the session, we know that any product previously presented to the user is not the target, so we may set the posterior on those products to 0.

7. SELECTING PRODUCTS FOR A SCREEN

After we have updated product scores, the system must choose which products to present in the next screen. This choice presents what has become known as an exploration-exploitation trade-off [13, 10, 8, 15], presenting a wide enough range of products that the user's choice exposes a lot about her preferences versus choosing products that are likely to appeal to the user immediately.

7.1 Thompson sampling

Thompson sampling [13] has been shown to work well for problems with an exploration-exploitation tradeoff, such as placing internet ads and recommending news articles, where the system only learns about the user's reaction to the content that was chosen [5, 3, 9]. In our system, the structure of the interaction with the user is richer, however Thompson sampling may be extended as follows. In the most basic version of our system, a screen of products is chosen to show to the user by repeatedly choosing a product at random, where the probability of choosing a product is equal to its posterior probability. Thompson sampling progressively enriches the options presented to the user for products that are likely to be of interest to the user. On the other hand, it continues to present the user with opportunities to express preferences for products that information up to a given point in time suggests might not be of interest. Choosing products with a probability equal to the probability that they are of interest strikes a delicate balance between these two: in the end, once a certain type of product can be eliminated with high probability, it becomes very unlikely to be presented to the user. Some theoretical analysis attests to the sensitivity of this balance [1, 7, 2].

7.2 Weighted k-medoids

Armed with a posterior probability distribution over user interests, our system may be configured to choose a screen in order to minimize a weighted average of the distances of the embeddings of the products in the catalog to the closest embedding of a product included in the screen. If we refer to all of the clicks up to a given point of time as c, the cost function of a screen S can be formulated as

$$\text{Cost}(S) = \sum_{x \in \text{Catalog}} \mathbf{Pr}(T = x|C = c) \min_{s \in S} d(x, s), \quad (2)$$

where $\mathbf{Pr}(T = x|C = c)$ is the posterior as defined in Section 6. A known algorithm may be applied; for example, weighted k-medoids is included in the R package WeightedCluster [12]. Using weighted k-medoids allows the algorithm to choose representatives of different kinds of products, but in a way that assigns higher priority to finding representatives of products that are likely to be of interest to the user. Since the chosen products are in the middle of regions of embedding space that are well-populated with potentially interesting content, they themselves are likely to be of interest. (Note also that members x of the catalog that are included in the screen S contribute 0 to the cost; this effect provides

a direct incentive to include x in S when $\mathbf{Pr}(T = x|C = c)$ is large.) Aside from speeding up the user's search, this also makes a screen after a click correspond more clearly to the click action, which provides satisfying immediate gratification to the user which may improve engagement. The potential disadvantage of this method is the time complexity: if the clusters are roughly equal-sized, n is the number of products, k is the number of clusters, the time required is roughly $\Theta(n^2/k)$.

7.3 Candidate reduction

Both of the above methods for choosing screens are improved (with respect to our metrics described in Section 8), when they are applied to a restricted subset of the catalog containing the highest scoring products. The fraction retained shrinks as a session proceeds. After i clicks, the fraction of the catalog kept is s^i, where the reduction factor s is a tunable parameter. If this reduction results in keeping fewer members of the catalog than there is space in the screen, the system simply outputs the top scoring products. Note that candidate reduction tilts the exploration/exploitation balance toward exploitation.

8. EVALUATION

We evaluate prospective changes to our system using techniques that trade off between cost, turnaround time, and meaningfulness in various ways.

8.1 Clicks to target with live judges

One evaluation method uses a game that we pay live judges to play. The judges are challenged to find an item, which is shown to them – this item is a randomly chosen product from the catalog. We then present screens of product images to them, chosen using our system, and count how many clicks they take to find the product.

If a judge cannot find the product after 20 clicks, we choose a new product for them. We pay users whether they find the product or not.

We have contracted with Spare5 (`https://spare5.com`) to run these tests.

8.2 Clicks to target with user simulations

We experiment more cheaply, with quicker turnaround, by using our user model to simulate users. As in the experiments described in Section 8.1, we choose a random target t, and average count clicks to target. When the system produces a screen S, we generate a simulated click by assuming that the user clicks on product $s \in S$ with probability $\mathbf{Pr}(s|S, t)$ using the model described in Section 4. If the simulation reaches a maximum amount of clicks, for example 20, when computing the average, we count it as the maximum (20).

These experiments can be used to assess changes to the system that do not involve changes to the embeddings, for example, how to choose elements of the screen to present to the user. This mode of evaluation may not be used to compare methods for training embeddings. To see why not, consider scaling up all the embeddings by a large constant. This would increase the simulated probability that the user clicks on the item closest to the target, and, generally, the probability of clicking on products would become more skewed toward products that are close to the target. Thus, as the embeddings become larger, the clicks to target will tend to decrease. This will be the case, even if the click distributions are not as skewed as such large embeddings would indicate. If we tried to use such embeddings in the field, the resulting inaccurate probability estimates would actually harm the system.

8.3 Log-loss with user model for evaluating embeddings

The ultimate test of the value of embeddings is observation of their effect on the qualitative behavior of the system. We may evaluate embeddings objectively using the experiments described in Section 8.1. We may obtain a cheaper preliminary evaluation of an embedding E using the user model P_E from Section 4 engendered by E, by calculating the average, over held-out test data consisting of a screen S, a target t and a click s, of $\log(1/\mathbf{Pr}_{P_E}(s|S, t))$. These tests can be used for fine-grained design decisions, like setting parameters for deep network training, choosing network architectures, et cetera. A diverse set of especially promising candidates may then be evaluated using the live user test described in Section 8.1.

8.4 Test data and simulating the introduction of new products

Immediately after a model is trained, it is applied with the same catalog that trained it. Thus, if we generate test data using the same catalog (which does not require generalization to images not used during training), we are faithfully reflecting conditions in the field.

After some time, however, new products are added. Thus, typically, in production, a catalog consists mostly of images that were seen during training, along with a few new images.

In order to produce test data with this property, we must simulate the addition of new products to the catalog. To do this, we choose a random subset N of products, and *exclude* them from the training data. When we simulate the system to choose t, S pairs to send to the judges, we behave as if products in N are not in the catalog. Then, to generate test data, we add the products in N back into the catalog. Thus, the members of N play a role in test data analogous to products added after training when the system is used in the field.

9. EXPERIMENTS

9.1 Spare5 experiments

As described in Section 8.1, we challenged users to find specific products of `SHOEme.ca`'s women's boots catalog of 3600 products. The system displayed 7 products per screen and we used a candidate reduction factor s of 0.6. Table 1 shows the results over 10k searches, for Thompson sampling and for weighted k-medoids. Recall that if a judge cannot find a product within 20 clicks, she is given a new task. When the system used Thompson sampling, judges succeeded slightly more often (though the difference is not statistically significant), though they required slightly more clicks to target when they did. To arrive at a single statistic to use to compare algorithms, we need to choose a number of clicks to "charge" an algorithm when the target is not found. One natural, but conservative, choice is 20. The fourth column, labeled PEN20, gives the average number of clicks when an algorithm is charged 20 clicks when the target is not found.

Algorithm	% found	ave. clicks when found	PEN20
Weighted k-medoids	$67.2 \pm 1.3\%$	8.54 ± 0.05	12.30
Thompson sampling	$69.2 \pm 1.3\%$	8.87 ± 0.05	12.30

Table 1: Results of a Spare5 test validating the system and comparing k-medoids with Thompson sampling.

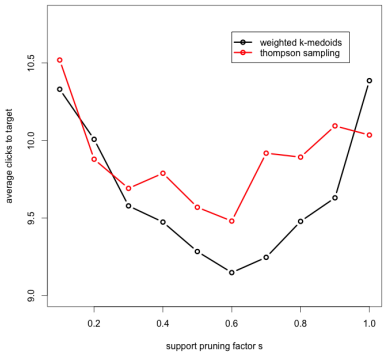

Figure 2: The effect of different values of the candidate reduction factor s on simulated clicks-to-target.

9.2 The effect of candidate reduction

In order to find the optimal setting for the candidate reduction factor s, we evaluated the clicks to target through simulations as described in Section 8.2. The experiments were performed on `SHOEme.ca`'s women's boots catalog of 4448 shoes, simulating 12 boots per screen. For each value of s, 10k searches were simulated with a maximum of 20 clicks per search. As Figure 2 shows, both for Thompson sampling and k-medoids, the value minimizing the simulated clicks to target is 0.6. K-medoids performs slightly better than Thompson sampling in simulations, but the difference was not observed with live users.

10. CONCLUSION

We have filed patent applications for the novel methods described in this paper.

10.1 The user interface

While the basic product provides an API through which we communicate product lists, and leaves the design of the interface to our partner, our company is working on its own user interface. A detailed description of this effort is beyond the scope of this paper.

10.2 Leveraging the prior

The production system assigns equal prior probability to each product in the catalog.

The most obvious way to use the prior is to assign higher prior probability to popular products. Doing this, however, would give rise to another exploration-exploitation tradeoff – the prior of a product affects its popularity. Setting the prior this way would reduce the incidence of "weird" products in Sentient Aware. On the one hand, this could make the system more aesthetically pleasing overall. On the other hand, it would dull the ability of the system to serve users with unusual tastes. Also, as the prior becomes more concentrated on popular products, our product becomes less differentiated from traditional offerings.

We also could expose the prior to our partners, allowing them to promote certain products. They may use this prior, for example, to assign lower priority to products that are nearly out of inventory and higher priority to products with excess inventory.

We also could use the prior to respond to brand preference on the part of the user. This way, a user could direct the system toward a certain kind of product without entirely excluding other products. To implement this, we need a way to go from a list of products to a prior. Many methods for estimating a probability distribution are available for this. We are experimenting with a method that uses a kernel density estimator (see [14]).

Another use of a prior would be to inform our interaction with the user by metadata that we can gather about the user. For example, it could be used to take account of the fact that, overall, tastes in San Francisco differ from tastes in Peoria. Incorporating this information using a prior with a system that adapts quickly to a given user's tastes, however, enables the system to react quickly to exceptions to geographical trends, providing a satisfying experience to users in Peoria with tastes like those more commonly found in San Francisco.

If we want to use priors for several of the purposes described about, a standard way to combine priors $P_1, ..., P_n$ is simply to multiply them and normalize so that the sum to 1.

10.3 When are we done?

Recall that our system never shows the same product twice. As expected, this practice improves the number of clicks required to find the target.

Consider, however, a scenario in which the user is looking for a green rain boot with daisies on it. If the catalog does not have any such products, and only has a few green rain boots, our system will produce increasingly focused screens for some time. Once it has displayed all of the green rain boots, however, it will start to display products that are less similar to the user's ideal product. This might make it appear that it has become lost. It should not be hard to detect this (as the posterior starts to become less peaked). In such cases, it might be useful to communicate with the user. Another option is to start to reintroduce previously displayed products at this stage.

10.4 Alternative user models

The target model that drives our design is of course an idealized working hypothesis. The limitations of this model are made clear when it is applied with increasingly large catalogs. As the catalog gets larger, it takes longer and longer for the user to find what she wants. This is because, no matter how similar another item in the catalog is to t, in

this model, the user will only be satisfied by t.

In an alternative model, for a given radius r, the user will accept any product with $d(s,t) \leq r$. A disadvantage of this model is the arbitrary choice of r. It also makes it more difficult to describe a game to users for training whose objective is clearly defined. In simulation studies, we have found that counting clicks to target is strongly associated with counting clicks until finding something within distance r of the target.

Another limitation of clicks-to-target is that it does not take account of the pleasure of browsing. In another model, we may assume that, instead of a single target, the user has an affinity function a, that assigns a non-negative value of each product that reflects how attractive this product is to the user. We then could assume that the user clicks on product s with probability proportional to $\exp(a(s))$. If we also assume that the user buys a product with probability proportional to $\exp(a(s))$, then a goal of tending to display products with large values of a simultaneously pursues sales, and pleasurable browsing. The use of such a model presents a challenge regarding training an embedding model. One reasonable working hypothesis may be that, for a model trained using triplet data, the value of $a(s)$ varies smoothly with the embedding of s: in other words, we would be assuming that similar products are similarly attractive. Training embeddings to reflect similarities, as in the original triplet network, would be compatible with this assumption.

Comparing systems based on different user models presents an evaluation challenge. For example, a method designed based on a clicks-to-target user model should be expected to produce better values in a clicks-to-target simulation, or with live clicks-to-target games. One way to compare would be to conduct A/B tests with a live system, comparing statistics such as the sales numbers, user engagement statistics, or indications of user interest such as add-to-cart.

11. REFERENCES

[1] S. Agrawal and N. Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *COLT*, 2012.

[2] S. Agrawal and N. Goyal. Thompson sampling for contextual bandits with linear payoffs. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 127–135, 2013.

[3] O. Chapelle and L. Li. An empirical evaluation of Thompson sampling. In *NIPS*, pages 2249–2257, 2011.

[4] I. J. Cox, M. L. Miller, T. P. Minka, T. V. Papathomas, and P. N. Yianilos. The Bayesian image retrieval system, PicHunter: theory, implementation, and psychophysical experiments. *Image Processing, IEEE Transactions on*, 9(1):20–37, 2000.

[5] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 13–20, 2010.

[6] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition - Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015, Proceedings*, pages 84–92, 2015.

[7] E. Kaufmann, N. Korda, and R. Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *Algorithmic Learning Theory*, pages 199–213. Springer, 2012.

[8] J. R. Krebs, A. Kacelnik, and P. Taylor. Test of optimal sampling by foraging great tits. *Nature*, 275(5675):27–31, 1978.

[9] B. C. May, N. Korda, A. Lee, and D. S. Leslie. Optimistic Bayesian sampling in contextual-bandit problems. *The Journal of Machine Learning Research*, 13(1):2069–2106, 2012.

[10] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[12] M. Studer. Weightedcluster library manual: A practical guide to creating typologies of trajectories in the social sciences with r. Technical report, LIVES Working Papers 24, 2013. DOI: 10.12682/lives.2296-1658.2013.24.

[13] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[14] J. W. Tukey. Exploratory data analysis. 1977.

[15] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989.