Recommending CouchSurfers: Machine Learning for Social Travel

Ron Sun **Tobi Baumgartner** **Tim Althoff** **Sergey Karayev**

Abstract

New technologies have made social interactions that were previously unthinkable commonplace. When traveling, a viable new option is to be hosted by a friendly resident of the destination, for free—to "couchsurf." When arranging such accomodation through a web interface, travelers face a daunting list of potential hosts, while hosts in popular destinations are inundated with requests for lodging. We postulate that some connections are more likely to happen than others, based on general desired compatibility between host and visitor and on personal host preferences. We present a system for predicting successful host-visitor matches, trained and evaluated on data from the most popular website for this purpose: couchsurfing.com. A re-ranking of visitor requests based on our system would be beneficial to the user experience on the website, as hosts would have to look through fewer requests to find a "couch surfer" they like. Furthermore, ranking the different hosts shown to the surfer according to the likelihood of a successful couch request would decrease the number of requests the surfer has to write.

1 Introduction

> Among those to whom I did not write "couch request" were "a travelling magician and professional fool" from New Mexico; a sixty-three-year-old gay semi-retired handyman in Pahoa, Hawaii, whose mission is "looking for more nudists" (there are plenty of "clothing optional" possibilities on CouchSurfing); another Hawaiian, this one describing himself as "just a guy who has three acres of land, living in a shipping container house"; a woman in Bozeman, Montana, who declared that her "home is oppression-free. Yay!" and also contains high-speed Wi-Fi; a thirty-one-year-old female "daydreamer" from Berkeley who loves pajamas; a Tarot-card reader in Marfa, Texas; a housewife in Charleston, South Carolina, who owns a pole-dancing studio called Dolphin Dance; a kite-surfing physician (again, Charleston) whose ambition is to start a flavored-envelope company; a freelance photographer in San Francisco who claims shes had hiccups every day for the past five years; a Savannah, Georgia, scientist who is also a "free man," doing "whatever I want, whenever I want"; a male in Antarctica who wishes to live as a female; a "lovetarian" who grew up in Doylestown, Pennsylvania; ... and eighty-four people in Brunei Darussalam.
>
> – Patricia Marx. *You're Welcome: Couch-surfing the globe*. New Yorker, April 16, 2012.[1]

Our new global interconnectedness has made social interactions that were previously unthinkable commonplace. Even a couple of decades ago a traveler had to either have a friend or pay for a hotel for lodging while traveling, but today people so inclined can stay for free at complete strangers'

[1] Retrieved from http://www.newyorker.com/reporting/2012/04/16/120416fa_fact_marx

residences, in locations that span all countries and continents. Such accomodations are arranged via special-purpose websites that let users willing to host travelers to list their apartments ("hosts"), and users wanting to with someone for free to request lodging from the hosts ("surfers"). Among these websites, by far the biggest is CouchSurfing (http://couchsurfing.org/), a website launched in 2004 and now boasting almost four million users.

There is a staggering variety of people on CouchSurfing. There are members in every country, speaking over three hundred different languages. The age distribution skews young: the average age is twenty-eight and a third are between 18 and 24. The website has facilitated over six million positively-reviewed visits, with "only a tiny fraction of one per cent negative."[2]

A fraternizing spirit is strong with most users of the site, as they are willing to host or guest with strangers in exchange for no money. All the same, we believe that people there are some user-match relationships that are more likely to be successful than others. Hosts are selective in accepting requests, and it should be possible to learn from their preferences.

When a surfer is looking for lodging, they do a search for available couches (or beds, or the floor, as the case may be) in the destination location. Usually, they are presented with an array of choices, often more than can quickly be looked through—or messaged. It would be beneficial to a searching surfer to have hosts that are available, agreeable, and likely to respond positively displayed at the top of the list. Similarly, hosts in popular locations are inundated with requests for accomodation. To save time, surfers that would be agreeable to them should be displayed at the top. An example of a host's inbox is presented in Figure 1.

In this work, we develop a system to automatically evaluate the couch requests that hosts get with respect to the likelihood of a successful and pleasant visit, and so make the CouchSurfing experience as convenient as possible. While we train and evaluate our approach from the perspective of the hosts, our solution to the problem—predicting the probability of a host accepting a surfer request—benefits surfers just as much, as their search results can be ranked according to the likelihood of acceptance to save them message-writing effort.

We detail our problem formulation further while evaluating related work in section 1.1. Next, we look at the structure of CouchSurfing data and user experience in section 2. In section 3, we formalize the problem and present our model. Our approach to extracting features from user data is presented in section 4. Finally, evaluation results are presented in section 5, and conclusions drawn in section 6.

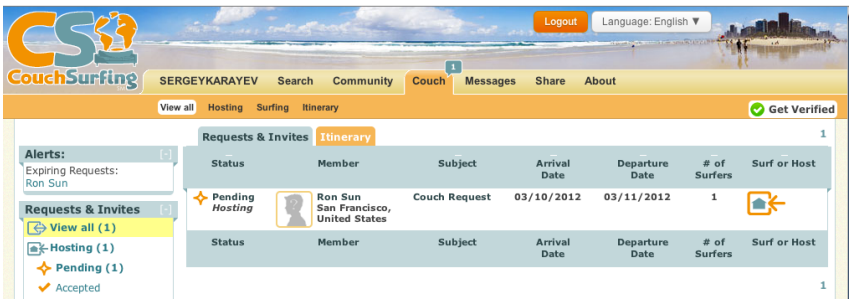


Figure 1: Example of a host's inbox view.

1.1 Related Work

CouchSurfing data has seen prior research user in the field of trust and reputation systems [1]. Reputation systems allow users to judge other user's trustworthiness and provides incentives for users to be honest. For this purpose, CouchSurfing.org employs a vouching system (declaring friends as trustworthy), a verification system (paying a small fee to get your name and address verified—the only revenue source for the website), and references (ratings and feedback) from other users.

In this work, we do not focus solely on reputation and trust, but rather on overall features of surfers and hosts that may signal likelihood of an accepted request.

[2] from the article quoted in the epigraph.

Recommender systems rely heavily on ratings contributed by users in order to predict other users preferences. Follow-up work has shown that public, identified ratings tend to be disproportionally positive if the ratee is another user who can reciprocate [2] (like on CouchSurfing). To the best of our knowledge this paper represents the first effort in using recommender systems and machine learning for social travel on platforms like CouchSurfing.

While we make use of ratings as a source of features for our prediction task, predicting the ratings themselves is not our goal. Rather, we aim to predict which couch request, among a group of competing ones, will be accepted by the host.

A similar problem arises in search engine optimization in a seminal paper on ranking SVMs [3]. Clickthrough data from users presented with a ranked list of results was used to improve the ranking function in a max-margin formulation. In our situation, hosts are indeed presented with a list of surfers (and surfers with a list of hosts), but the time of presentation as well as the clickthrough data is unknown to us.

We can, however, infer rough presentation times from the request and decision times, and we have the resulting accept or reject decision of the host. In this way, our problem is of classification and not ranking: given an (inferred) competitor set of couch requests, we predict whether each request was accepted or rejected, with the constraint that at most one request could have been accepted.

There are two parts to this prediction: a global "attraction" model that explains in general why some hosts prefer some users to others; and a host-specific personal model that allows additional flexibility. Personalizing classifiers in this way leads to a very large number of parameters that often is infeasible to keep in memory. To enable for personalization in a large-scale setting, the "hashing trick" has been proposed where either features or parameters are hashed into an array of predefined size. This potentially allows features or parameters to collide, but it has been successfully applied in personalized spam classification [4] and investigated theoretically [5]. We use the hashing trick to store personalized host parameters.

Another related strand of research is in predicting friendship connections, which is considered in [6] using joint information in existing friendships and common interests based on the hypothesis of homophily. The hypothesis is that people with similar interests attract each other: a premise that our model has sufficient power to express, but is not biased toward. Similarly, we use the interests of hosts and surfers expressed in free form text on their personal profiles to estimate the likelihood of an accepted request. Some evaluation schemes in [6] are also relevant to our problem formulation.

For learning the parameters of our model, we use a parallel Stochastic Gradient Descent (SGD) algorithm that distributes random subsets of the data onto multiple machines and uses fast lock-free updates on a local copy of the parameters [7]. We modify the algorithm in that after a certain number of iterations, the results are averaged and re-broadcast, for additional robustness.

As part of our featurization function, we extract "interests" from free-form text of the user profiles. In this challenging task we are guided by [8], who use a graph structure assembled from overlapping user interests, and [9], who deal with semantic annotation: essentially mapping a document to its ontology entities (almost like topic modeling). We learn the graph based on a novel normalization scheme, and use clusters found in the structure as features in our model.

```
Your featurization approach is quite hacky.
```

2 CouchSurfing Experience and Data

CouchSurfing.org is a worldwide network that envisions "a world where everyone can explore and create meaningful connections with the people and places they encounter."[3] More than four million users participate in the site through hosting travelers on their own couch, "surfing" on other users' couches when traveling in a foreign city, or finding and participating in local CouchSurfing meetups in their area [1].

CouchSurfers have been to 252 different countries and speak 366 unique languages.[4] Being a part of a social travel community, each user presents themselves in their user profile (see Figure 2 for

[3] Retrieved from http://www.couchsurfing.org/about.html/vision

[4] Retrieved from http://www.couchsurfing.org/statistics.html

an example). These profiles contain rich information about the user, their CouchSurfing experience, what languages they speak, their interest, which countries they have traveled to, and much more.

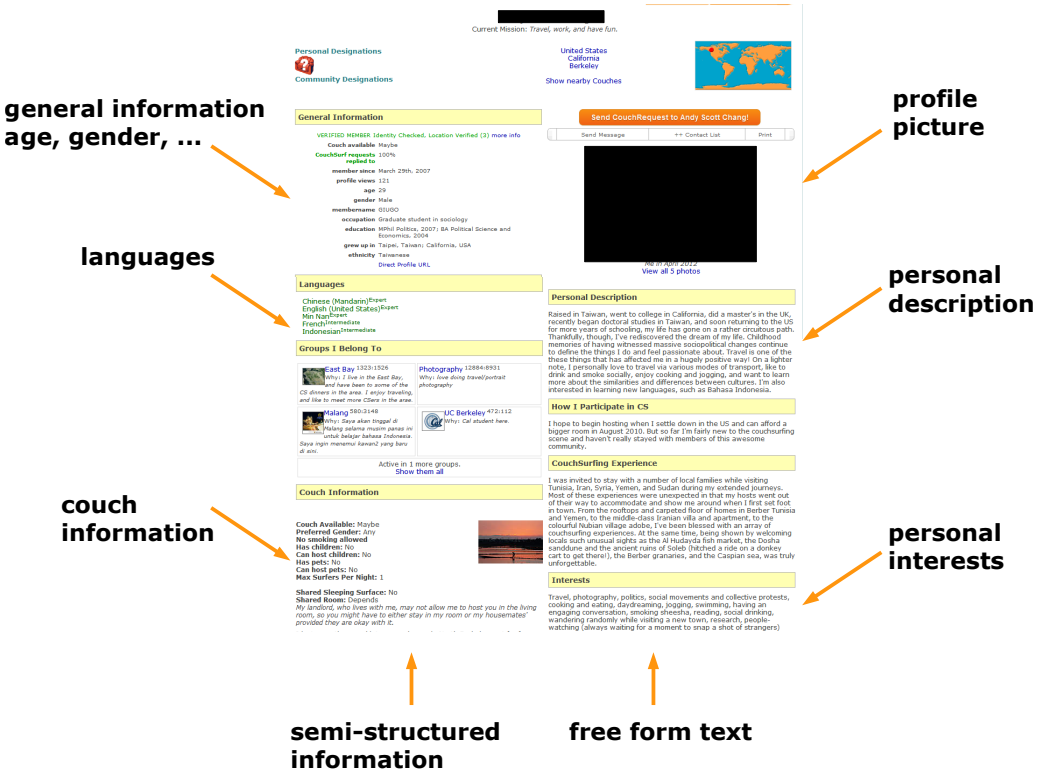


Figure 2: User profiles on CouchSurfing.org contain lots of valuable information to estimate host and user preferences. While the left column is comprised of (semi-)structured information like age, gender, and languages spoken, whereas the right column solely consists of free textual descriptions.

While CouchSurfing has become more popular, it has not necessarily become easier to find a place to stay for surfers or find the right surfer to host. Potential hosts often get dozens of couch requests a day because the surfers need to send out more requests due to the increased competition between surfers. Particularly, hosts in popular locations are inundated with requests for accomodation (e.g. in New York City, Paris, London, Berlin, and Istanbul).

To achieve a good user experience, the acceptance rate of couchrequests needs to be high, i.e. the number of requests needed to eventually find a place should be minimized. On the other hand, hosts need assistance in deciding which surfer they would like to host. In this paper, we tackle both of these problems but focus on the latter (based on the expressed need of developers at CouchSurfing[5]).

We estimate the probability that a user's couchrequest to a specific host is going to be successful based on the profiles of both user and host and the host's preferences. The couch requests for a given host can then be re-ranked such that the best candidates are at the top (according to the hosts personal preferences), and the hosts can be re-ranked for a surfer's search request.

3 Problem Formalization and Prediction Model

Now we formalize the problem and present our model. All notation is also presented in Table 1 for reference.

[5]Private communication

Table 1: Summary of our notation.

Symbol	Meaning
$h \in \mathcal{H}$	Host, from a set of all users who have received at least K couch requests.
$\mathcal{S}_h$	Set of all competitor sets S for host h.
$S \in \mathcal{S}_h$	Competitor set for host h, composed of surfers who have made a couch request to h at roughly the same time.
$\Phi(s, h, c)$	Featurization function mapping a couch request c from surfer s to host h to $\mathbb{R}^D$.
θ,r	Global parameters: weights for the couch request features, and a "reject-all" parameter.
θ_h,r_h	Corresponding host-specific parameters.

From the CouchSurfing user database, we form a set of *hosts* $\mathcal{H}$ such that every $h \in \mathcal{H}$ has hosted at least K persons since signing up for a user account. We initially set $K = 1$; a higher setting may reduce the amount of noise in the set of hosts.

For every host h, there is at least one *competitor set* S of at least K *surfers*, where each surfer $s \in S$ has made a couch request to h. A competitor set corresponds to a decision made by the host to either host one of the s in the set, or to reject all of them. The composition of this set is inferred from data based on the timing of couch requests and host decisions. This is discussed in section 3.1.

In fact, h may have multiple such competitor sets. We refer to the set of competitor sets for host h by $\mathcal{S}_h$.

Given that the host solves a dynamic program you could probably also use the information as to how booked the host already is.

3.1 Competitor Sets

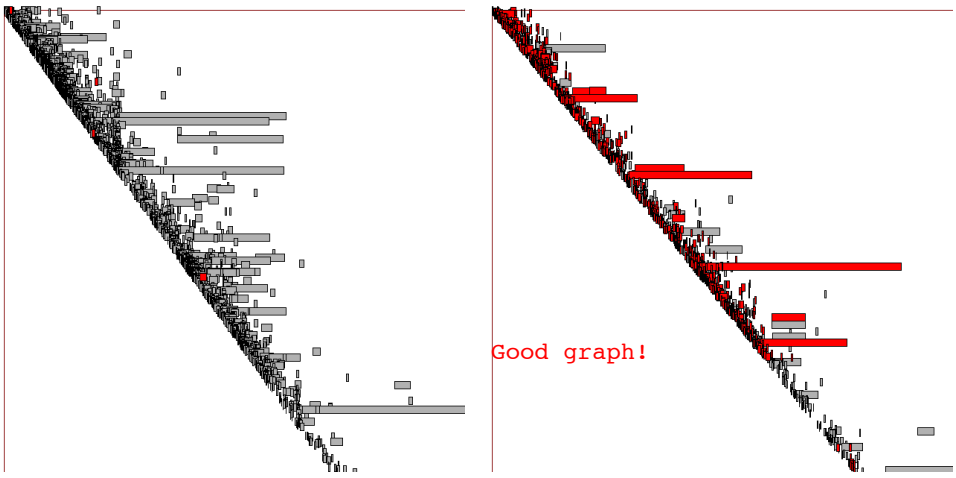
The data we received from CouchSurfing does not allow us to know exactly when a host has looked at a couch request, but it does have timestamps for every request and every decision made by a host. To form sets of "competing" requests, we use a heuristic procedure to scan through all requests made to a particular host, and split them into groups based on their timing.

In the following, requests carry the same index as the user/surfer that invoked them, i.e., surfer s_i sends request $c_{i,j}$ to host h_j. In the following examples, h is fixed, so we just denote requests as c_i.

We assume a session-based behavior for the user h, which means he sits down in front of his computer and checks all new request to his couch every once in a while. There will be several different surfers that want to stay with him during different times. Now within a certain concetration span, h will read through all the requests (besides the basic infos about age, sex and location, a potential surfer s_i can also specify various details about his itinerary and whatever else will help him convince h). While reading through the requests h decides on the fly which of them to accept or decline. Based on this notion we chunk together request by their *request modification date* (rmd), the date h responded uppon them with *accept*, *declined* or *maybe*. More specifically, we order all request for each host by their rmd and perform a mean-shift clustering with a bandwidth of the session length, e.g. 30 min.

As described later we now use the winner in a competitor set as a positive sample and all other requests as negatives. Since a ranking between two winners in the same set cannot be done reasonably - we have no notion of who would be better in this case - and to match our model optimally, we split sets with several winners into multiple sets with one winner each. With that we expand the number of competitor sets from 4.5 million to 5.5 million.

In figure 3 we plot the requests that are sent to one host. The gray/red boxes stand for one request; Their width describes the period of time they want to stay. The vertical position stands for the point in time when this request was filed. We draw a box in read if the host accepted this request and in gray if it is declined. As can easily be seen, there are some users (cf. figure 3a) that receive an overwhelming amount of requests and others that accept most of the requests they receive.



(a) Requested (b) Accepted

Figure 3: Couchrequest for two extreme users

3.2 Prediction Model

		w/o reject	w/ reject
h_1	r	-	0.2
	s_1	**0.5**	**0.4**
	s_2	0.5	0.4
h_2	r	-	0.6
	s_1	**0.75**	**0.3**
	s_2	0.25	0.1

Figure 4: Example of a case where modeling the reject option results in vastly different recommendation behavior. Consider surfer s_1's perspective: without modeling the reject option, h_2 looks more likely to accept s_1. When modeling the reject option, however, the probabilities change and h_1 is now more likely to accept s_1, due to the ability to model how picky h_2 actually is.

We represent the probability that a host h chooses surfer s_k out of a competitor set S with a logistic model

Make reject depend on context, too.

$$p(s_k|h,S) = \frac{\exp(\theta^T \Phi(s_k,h,c)}{\sum_{s_j \in S} \exp(\theta^T \Phi(s_j,h,c)) + \exp(r)} \quad (1)$$

where $\Phi(s_k,h,c)$ is the output of the featurization function $\Phi : (s,h,c) \mapsto \mathbb{R}^D$ and θ is the D-dimensional parameter vector we learn. The parameter c represents the context of the couch request, such as time, and will henceforth be omitted from notation but implictly included. We also implictly model a bias term by appending 1 to the representation $\Phi(s_k,h)$.

Accordingly, we model the probability that h rejects all $s_k \in S$ as

$$p(\text{reject}|h, S) = \frac{\exp(r)}{\sum_{s_j \in S} \exp(\theta^T \Phi(s_j, h)) + \exp(r)} \quad (2)$$

In this formulation, each surfer s_k receives a score $\exp(\theta^T \Phi(s_k, h))$ modeling the natural competition between multiple surfers that requested to stay with the host. In the end, we predict that the surfer with the highest score wins, or that everybody gets rejected if the "reject score" $\exp(r)$ is largest. Note, however, that this is not exactly the standard multinomial logistic regression model since in our case we have the same parameters but we have different features for the different "classes" (i.e. surfers).

It may at first seem that we are essentially simply ranking the surfers, and so a reject option is not needed. In fact, we are modeling the host decision process, and the reject option is very important. As Figure 4 demonstrates, the reject options enables us to model the "pickiness" of a host. This allows us greater accuracy in predicting the decision outcomes for a competitor set, and allows CouchSurfing to rank potential hosts returned in a user search according to their actual likelihood of acceptance.

The parameters θ for this model should capture what hosts prefer in surfers and their request, i.e. their language, nationality, age, gender etc. Because the feature $\Phi(s_k, h)$ depends on both the surfer and the host we also model that i.e. Americans like to host surfers from certain countries and that hosts generally prefer to have a language in common with the couchsurfer. Further details are given in section 4.

You could also learn the preferences of the surfer in the same fashion. This will then turn into a recommender system.

3.2.1 Host preference personalization

However, these preferences may be very different from host to host. Also, some hosts might be more picky than others rejecting almost all requests and we should be able to capure that, too. Therefore, we introduce host-specific parameters θ_h and r_h to model these effects. Because we have millions of hosts storing all these parameters explicitly would require tens of gigabytes of memory.

Since this is infeasible we make use of the "hashing trick" to hash the parameters into an array of predefined size (allowing for collisions between parameters). Thereby, we can control how much memory we want to allocate for personalization. This trick has been successfully applied e.g. for spam classification [4].

Including personalized parameters leads to the following updated model:

$$p(s_k|h, S) = \frac{\exp((\theta + \theta_h)^T \Phi(s_k, h))}{\sum_{s_j \in S} \exp((\theta + \theta_h)^T \Phi(s_j, h)) + \exp(r + r_h)} \quad (3)$$

$$p(\text{reject}|h, S) = \frac{\exp(r + r_h)}{\sum_{s_j \in S} \exp((\theta + \theta_h)^T \Phi(s_j, h)) + \exp(r + r_h)} \quad (4)$$

3.2.2 Learning the parameters

As an error measure, we use the negative log-likelihood of the data with l_2-penalty on the parameters for regularization. In our Stochastic Gradient Descent (SGD), we randomly sample a host h and a competitorset $S = \{s_1, \ldots, s_n\}$. There are two cases: either there is one surfer s_{k^*} that got accepted, or everybody got rejected. We obtain the following update equations for SGD where λ_1, λ_2 are regularization parameters and η is the learning rate

1. Case: Some surfer s_{k^*} gets chosen.

$$\theta \leftarrow (1 - \eta\lambda_1)\theta - \eta(\sum_{j \in S_n} p(s_j|h, S)\Phi(s_j, h_n) - \Phi(s_{k^*}, h_n)) \quad (5)$$

$$r \leftarrow (1 - \eta\lambda_2)r - \eta p(\text{reject}|h_n, S_n) \quad (6)$$

2. Case: No surfer gets chosen (reject).

$$\theta \leftarrow (1 - \eta\lambda_1)\theta - \eta(\sum_{j \in S_n} p(s_k|h, S)\Phi(s_j, h_n)) \quad (7)$$

$$r \leftarrow (1 - \eta\lambda_2)r - \eta(p(\text{reject}|h_n, S_n) - 1) \quad (8)$$

The update equations for θ_h and r_h are the same as the ones for θ and r, respectively, but they are only updated for the current host h.

These update equations make intuitive sense, as for θ, we essentially increase the contribution from the features of the winner and decrease it for all the losers always according to their probability of being chosen. This means that if the model did a bad prediction we update our parameters more than normally. And for r, the story is similar in the sense that we increase the parameter value whenever everyone in the competitor set actually got rejected, and decrease it when we have a winner.

To scale the learning computation, we follow [7]'s SimuParallelSGD algorithm, and distribute random T-sized subsets of the data to m different machines. On each worker, SGD is used to update the parameters, sampling data in random order. We modify the algorithm by repeating the above in a loop, each time averaging the per-machine parameter vectors and re-distributing them back to all workers. The main idea of the algorithm is that in a large-scale learning situation such as concerns the CouchSurfing database, one can consider the present data to be a subset of a vritually infinite set, and so drawing with replacement and without replacement can both be simulated by shuffling the data and accessing it sequentially.

How large is your database? I would have expected that good C+ code would do the trick, too rather than parallelizing. That said it's great that you did it.

4 Features

The featurization function $\Phi(s, h, r)$ maps properties of a couch request r from surfer s to host h to a real-dimensional D-vector. The properties we care about include structured personal information such as age, gender, languages spoken, countries lived in, countries traveled to, and so on. Additionally, we would like to use user interests, extracted from free form text on their profiles, and their status and activity on the website, e.g. the number of references, number of friends, or message response rate.

You should smooth. E.g. 25 and 26 year olds aren't very different.

We developed a general approach for dealing with a wide variety of features. For each feature, we construct a histogram whose boundaries are determined by the inverse distribution of values in the training set. Using this mapping from value to bin we vectorize each feature by mapping it to a binary vector with the constraint of summing to 1 (e.g. $age = 27$ to $b = [0, 0, 0, 1, 0, \dots]^T$). Since our feature $\Phi(s, h)$ is based on both the surfer s and the host h, the final feature includes this binary vector for each of them, as well as the outer product between the two, $b_s b_h^T$.

The reason for the outer product feature is to capture potential preference combinations. For example, we are able to model that hosts between 21 and 25 years old prefer surfers of the same age, while hosts over 60 don't have a particular preferences in this regard (as an example). Other example inlude capturing affinities between genders, countries, spoken languages, certain interest groups, etc.

In addition to these cross-product preference features, we include additional unary features, such as the number of languages in common, and features relating to reputation. The final feature vector is quite sparse.

4.1 User interests

We also extract "interests" features from free-form user profile text as additional signal to the feature vector. Guided by [8, 10, 9], we form a graph of profile terms found in user profiles. The edge strength between two terms is proportional to the number of times the terms co-occur in user profiles. This measure is of course biased toward frequently-occuring terms. We use a normalization scheme that accounts for this, and the edge strength between w_1 and w_2 is determined as

$$\frac{P(w_1, w_2)}{2}\left(\frac{1}{P(w_1)} + \frac{1}{P(w_2)}\right) \tag{9}$$

On this graph, we are able to find clusters of common terms, which we refer to as "interests." We use an algorithm from [11], implemented in open-source software[6] An example of such a clustered graph is given in Figure 5.

[6] Gephi at http://gephi.org, based on GraphViz.

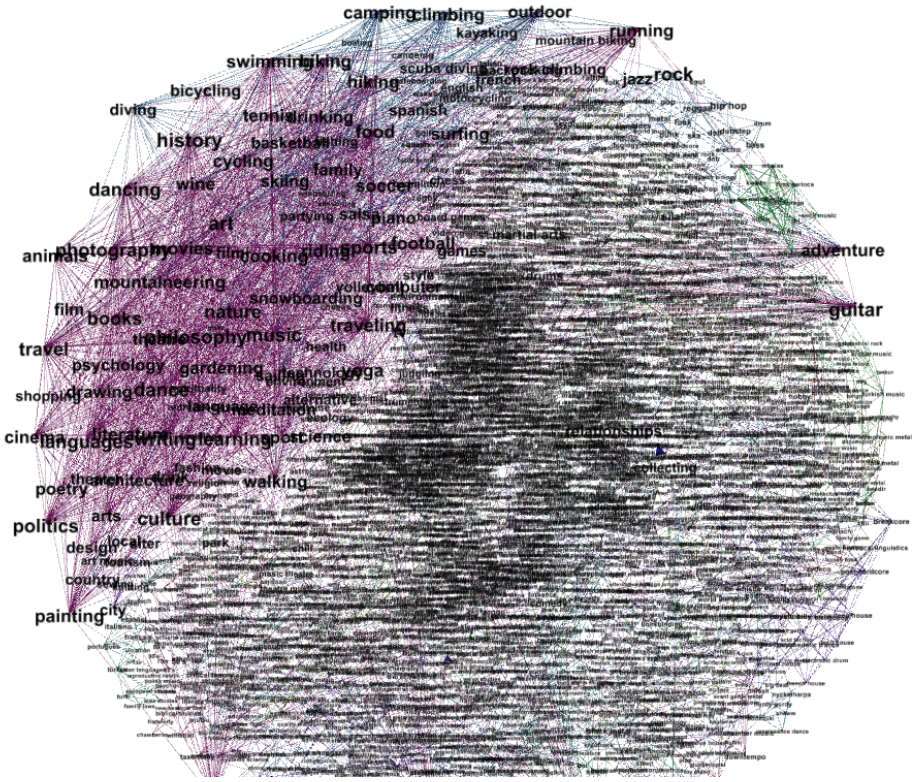


Figure 5: Graph of terms occuring in free-form user profile text, with "interest" clustered in color.

We vectorize a user's interests to a binary vector i that has no constraint of summing to 1, e.g. a user expressing clusters 2 and 4 out of $C = 5$ total clusters would have $i = [0, 0, 1, 0, 1]$. We pick the number of clusters C by hand, and include the cross-product feature (defined as before) into the final feature $\Phi(s, h)$.

5 Evaluation

The CouchSurfing dataset was made available to the authors in the form of an anonymized MySQL database dump.

Due to the communicated need of CouchSurfing developers, we focused our evaluation on the host perspective, where accuracy in predicting rejects is not as important as accuracy in predicting a successful visitor, and assistance is particularly important when the host faces a lot of couch requests at once. Therefore, we look at all competitor sets that have a winner, grouped by cardinality.

We compare our system against a random baseline as well as several rule-based approaches that may form the system currently deployed at Couchsurfing. These baselines present simple feature-based rules, such as ordering by the number of references or friends. We call these single-feature baselines. Although the exact ranking model currently deployed at CouchSurfing was not shared with us, we were told that it is a simple rule-based approach that most likely involves the parameters mentioned above.

The specific baselines were:

- *Random* Baseline that predicts a random element from the competitor set
- *Priority1/2* Baseline that CouchSurfing presumably uses to rank surfers and hosts on their website
- *Vouched* The number of people that vouched for a given user

- *References/to* References given to and received from other users
- *Friend links* Number of friends on CouchSurfing

Use all competitors as features!

We use two different measures to evaluate performance: Prediction Accuracy (PA), and Average Normalized Winner Rank (ANWR). Prediction Accuracy is the fraction of correctly predicted winners from several competitor sets. Average Normalized Winner Rank is a more general measure that captures what position the actual winner has in the computed ranking. We normalize the rank to be on $[0, 1]$. Note that a prediction only counts as a positive towards PA if it occured at the first place in the ranking. The ANWR is a less strict evaluation metric in this case, but it may better represent the actual use case we are developing for.

We compute these two measure on the CouchSurfing Data that has 11 million couch request, grouped into 5.5 million competitor sets. We split these competitor sets into a training, validation, and test set (at a 60/20/20% split). To ensure that we do not look ahead into the future to make predictions easier, we split the data chronologically.

Note that the state of the system at the time of feature computation lies "in the future" relative to the request being represented. This has certain effects, described below.

For a summary of the results see Figure 6 which shows PA and ANWR on competitor sets of a certain minimum cardinality. One can observe that we consistently outperform the baselines by several percent. For larger competitor sets, the recommendation or ranking task becomes harder and performance drops. Especially in this regime, our algorithm supports hosts better in choosing the right couch surfer to accept.

Due to the normalization the ANWR error metric stays roughly constant as cardinality of the set increases. The average rank for the random baseline is 0.5 ("the middle"). Interestingly, some of the single-feature baselines do worse than the random baseline (e.g. the number of people that have vouched to you). The number of friends and the number of references performs better than random. Using our system, on average, a host needs to browse 39% of the list compared to 46% for the the second best approach.

A common measure is the expected reciprocal rank. That is, 1/position.

5.1 Discussion

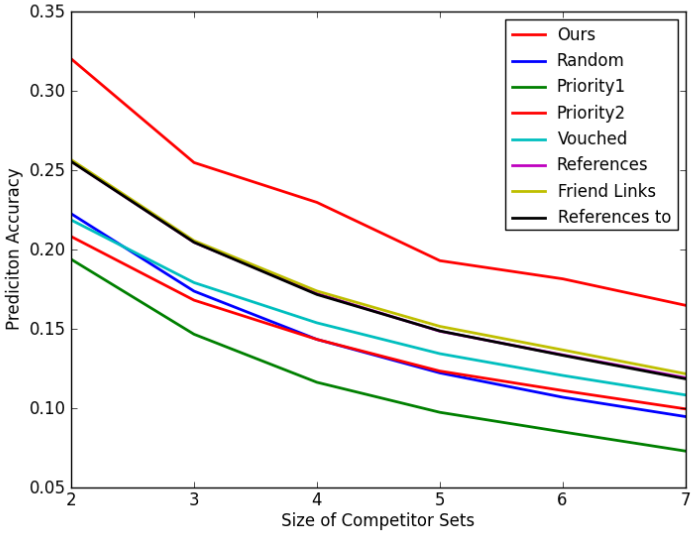
Why is this such a hard problem? First of all, notice that estimating probabilities including the possibility of rejection is much harder than just ranking of candidates due to the fact that rejectance might have nothing to do with the candidates themself or the hosts preference but may simply be dependent on the host's availability. This availability is hard to to infer automatically from the user's profile. For us it was impossible to capture this kind of time-dependent information because we were only supplied with the current state of the CouchSurfing database.

Furthermore, note that this also means that we cannot know e.g. when two users became friends on the network. This can lead to interfering behavior. Assume that we know that host h accepted surfer s from a competitor set S and we know that they are friends. Our model infers that it this friendship might be *the* critical and discriminative feature. However, h and s might not have been friends before the accepted couchrequest but only after they spent some time with each other. Similarly, our current model does not capture other temporal relationships such as whether host and surfer had been in contact before the actual request. We hypothesize that accurate modeling of this temporal structure in the data should improve performance significantly.

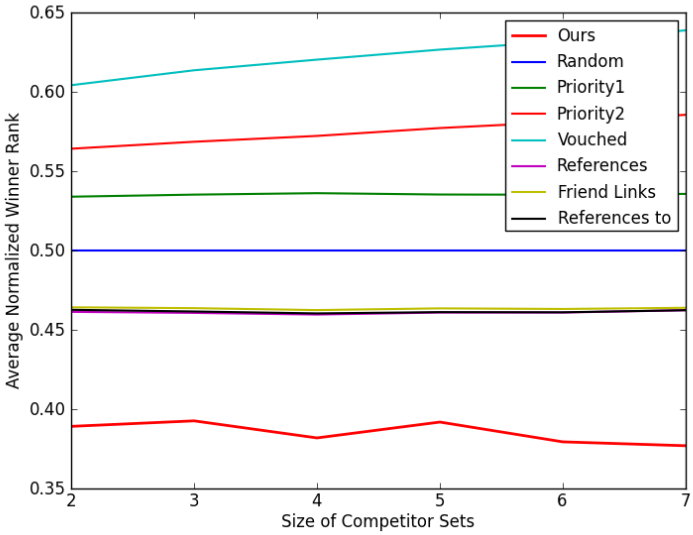
Discuss how to rank for surfers, too. You want to maximize chances of both couch owners and surfers. For the latter it's bad if they apply and don't get a chance. You can actually do this at every step by checking what will increase the overall number of couches accepted.

6 Conclusion

To the best our knowledge this paper represents the first effort in using recommender systems and machine learning for social travel on platforms like CouchSurfing. Our proposed system recommends surfers to hosts and vice versa in order to maximize the acceptance rate of couchrequests. This benefits both surfer and hosts: the surfer has to write less requests in order to find a place to stay, and the host has to review less requests to find a surfer he would like to host. We model the acceptance probability using a rich feature set including personal information like age, gender, languages spoken, countries lived in, countries traveled to, etc., their interests, and their status and activity on Couchsurfing.org which includes e.g. the number of references, the number of friends, or



(a) Prediction Accuracy (higher is better)



(b) Average Winner Rank (lower is better)

Figure 6: Our evaluations: prediction accuracy and average winner rank. In both cases, we do better than all baselines.

the response rate in the internal messaging system. We learn what hosts prefer in couch surfers and which combinations of attributes increase their chance of getting accepted. Furthermore, we allow for host-specific personalization using the hashing trick since different hosts have different preferences. We have shown that our proposed model can more accurately predict which surfer is most likely to get accepted by a given host than a random baseline and baselines similar to the currently deployed ranking system at CouchSurfing.

Future work includes an evaluation from the surfer's side, i.e. whether we can accuratly predict by which host he is going to be accepted. Further, we would like to investigate the benefits of modeling more dependencies across user properties and preferences (possibly using kernels), i.e. whether age

influences preferences in gender or whether a host that wants to travel to a certain country likes to host surfers from that country. We plan to evaluate whether our recommender system leads to a higher acceptance rate in the deployed system at CouchSurfing. Using this system we also want to explore more the temporal structure in the data better, i.e. whether both parties communicated before the request, were already friends, or whether the couch was already booked for the requested time slot or still available etc. The latter information could be used to filter the training data because the only reason for the rejection might have been the unavailalability of the host or his couch.

References

[1] D. Lauterbach, H. Truong, T. Shah, and L. Adamic, "Surfing a Web of Trust: Reputation and Reciprocity on CouchSurfing.com," *2009 International Conference on Computational Science and Engineering*, pp. 346–353, 2009. 2, 3

[2] C.-y. Teng, D. Lauterbach, L. A. Adamic, and A. Arbor, "I rate you. You rate me. Should we do so publicly?," in *Proceedings of the 3rd conference on Online social networks*, 2010. 3

[3] T. Joachims, "Optimizing Search Engines using Clickthrough Data," in *SIGKDD*, 2002. 3

[4] J. Attenberg, K. Weinberger, A. Dasgupta, A. Smola, and M. Zinkevich, "Collaborative Email-Spam Filtering with the Hashing-Trick," in *CEAS*, pp. 1–4, 2009. 3, 7

[5] K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. Smola, "Feature Hashing for Large Scale Multitask Learning." 2012. 3

[6] S. H. Yang, B. Long, and A. Smola, "Like like alike Joint Friendship and Interest Propagation in Social Networks," in *WWW*, 2011. 3

[7] M. A. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized Stochastic Gradient Descent," in *NIPS*, 2010. 3, 8

[8] H. Liu and P. Maes, "InterestMap: Harvesting Social Network Profiles for Recommendations," in *IUT*, 2005. 3, 8

[9] I. Cantador, P. Castells, and A. Bellogín, "An Enhanced Semantic Layer for Hybrid Recommender Systems: Application to News Recommendation," *Int. J. Semantic Web Inf. Syst*, vol. 7, pp. 44–78, 2011. 3, 8

[10] H. Liu, P. Maes, and G. Davenport, "Unraveling the Taste Fabric of Social Networks," *International Journal on Semantic Web and Information Systems*, vol. 2, no. 1, pp. 42–71, 2006. 8

[11] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, p. P10008, Oct. 2008. 8