Predicting Negative CouchSurfing Experiences using Local Information

Jan Overgoor
Stanford University
overgoor@stanford.edu

Ellery Wulczyn
Stanford University
ewulczyn@stanford.edu

Abstract

We attempt to predict negative references in the CouchSurfing (CS) social network using an SVM classifier. The low frequency of negative references, however, makes prediction very difficult. We attempt to achieve a classifier with a high F_1 score by training on data sets with varying distributions of positive and negative examples and by manipulating the decision boundary. We also explore how trust propagates through the CS network in order to design a set of features that represent personalized evaluations of other users. Our model of these local evaluations increases classification performance.

1 Introduction

CouchSurfing[1] (CS), is a social networking site that allows travelers to find temporary hosts on their journeys. Users of the site maintain a public profile that contains certain personal information and a history of references. After a hospitality exchange users leave each other textual references, which are tagged as either positive, neutral or negative. This reputation system allows people that have never met before to evaluate each other, and revealing misbehavior. The reference system is, however, not a fully reliable representation of the sentiment between users. Of the 6.1M reference in the system, only 98K are non-positive (80K are neutral and 18K are negative). (Lauterbach et al., 2009) showed that these numbers give a more positive image than what actually happens. Not every one leaves a reference after an experience and if they do, they are strongly biassed towards a positive one in fear of reciprocity.

This also means that a negative reference, if it occurs, carries a stronger weight and probably represents a particularly negative experience. We attempt to train an SVM to predict negative experiences based on each user's personal information and position in the CS social graph. We first describe the data set and the features we used. After discussing the results, we present an alternative method that favors information that comes from trusted sources.

[1] http://www.couchsurfing.org/

2 Data & Features

The complete dataset consists of 3.4M user profiles and 8.7M connections between users. A connection consists of a friendship link (7.5M) and/or a reference (6.1M). Friendship links are accompanied by a value between 1 ("*haven't met*") and 7 ("*best friend*"), as well as a confidential assessment of how much the users trust each other. A legend of these values is displayed in Table 1.

Value	Semantics
1	*I don't know them well enough to decide*
2	*I don't trust this person*
3	*I trust this person somewhat*
4	*I generally trust this person*
5	*I highly trust this person*
6	*I would trust this person with my life*

Table 1: Legend of trust values

In our classification task, a training example $(x^{(i)}, y^{(i)})$ represents a directed experience between two users, as evaluated by one of the users. $y^{(i)}$ is the evaluation of user T by user S, where a positive example (+1) refers to a negative rating and a negative example (-1) refers to a non-negative rating (either positive or neutral). We applied this inversion because we want to spot negative ratings. The feature vector $x^{(i)}$, consists of three parts: one set of features for both users and a short amount of comparison features.

A single user's feature vector consists of per-

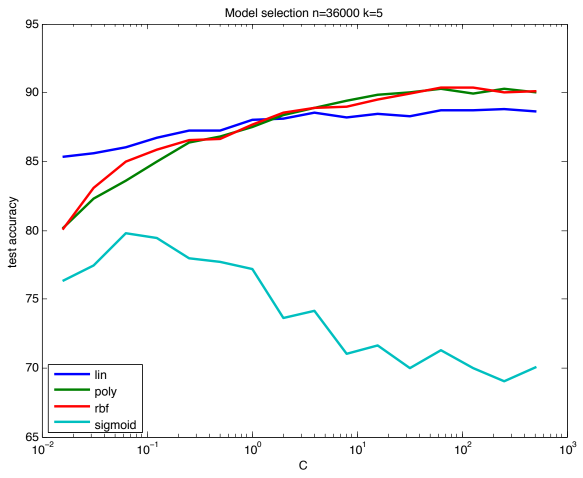


Figure 1: Model selection on different kernels and $c \in [2^{-6} : 2^{9}]$

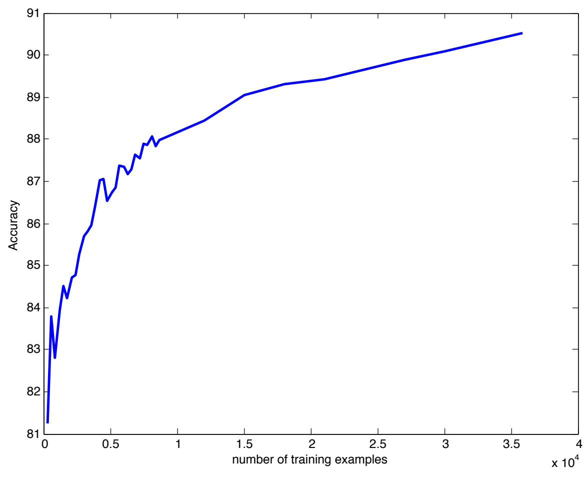


Figure 2: Quantity analysis with RBF kernel and $c = 256$

sonal attributes (e.g. age, sex and location), evaluation *by* the CS community (represented by distribution of friend scores, trust levels and number of references received) and an evaluation *of* the CS community (the same attributes, but *given* instead of *received*). The comparison feature set contains the difference between users in sex, age, location of origin and experience on the site.

In constructing our training examples we discard any information from after the reference date so as to simulate real prediction conditions, where there is no post facto data available[2].

3 Method & Results

We based our experimental set-up on the libsvm SVM library (Chang and Lin, 2011), which we accessed with our own Python bindings. We implemented a caching system for the dot product computation and made the decision boundary adjustable. Our initial investigations concerned balanced training and test sets, with an equal number of positive and negative examples. We used 5-fold cross-validation for every result in this paper.

We first performed model selection to select the best kernel and error cost parameter c. The results (displayed in Figure 1) show that, above $c = 128$, there is little difference between the polynomial (with $d = 3$) and RBF kernels. A comparison in computation time made us choose the RBF kernel and $c = 256$.

[2]Without this precaution, it is quite easy to produce very accurate classifiers, which take advantage of information that accumulated in reaction to the negative reference and, of course, the negative reference itself!

We also looked the learning rate as a function on the number of training examples, because the size of the training set has a big effect on our running time. Figure 2 shows that from about 15K training examples the difference in accuracy is relatively small, so we decided on using that number for the rest of our queries. Interestingly, it seems that learning has not stopped yet with the 36K training examples that we gave it, but the number of training examples is bound by the number of negative references in the data set.

A third preliminary test we did was ablative analysis. Because of our large amount of features ($n = 110$) and their conceptual proximity, we chose to do ablative analysis on feature sets as opposed to individual features. The results are displayed in Table 2. “Attributes u” includes both the personal attributes and the evaluation behavior of u. The most striking result its that the 9 comparison features have the biggest impact on the performance, but in general the relative impact of the feature sets is not significant. We considered using feature selection to reduce noise, but the run time of backward search proved prohibitive. Our ablative analysis does not hint at feature selection making a big difference, so we decided against it.

The overall performance of our system (an accuracy of above 90%) seems like a great result. But in reality, the ratio of positive and negative examples is heavily biased: only 0.3% or instances in the total data set are negative (so positive examples). Employing our classifier to detect negative references would behave gravely trigger happy (precision = 0.015). In order to improve perfor-

Feature Set	Accuracy
Attributes S	87.258
Evaluation S	86.004
Attributes T	87.672
Evaluation T	87.072
Comparison	85.772

Table 2: Ablative analysis: accuracy when leaving a feature set out

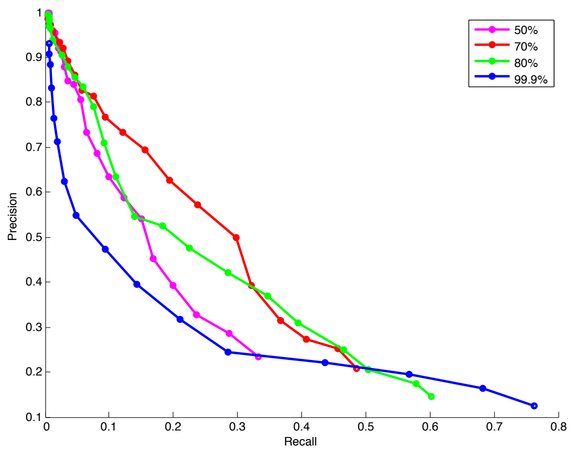


Figure 3: Precision/Recall curves for different training sets

mance on a skewed test set (50 positive examples and 9950 negative examples) we tried variations on two aspects: the decision boundary (t) and the skewness of the training set. Figure 3 shows how the skewness of the training set affects the Precision/Recall curve (made by varying the prediction threshold) and Figure 4 shows the performance as expressed in F_1 scores. The different training sets are expressed by the percentage of *negative* training examples, so 99.9% refers to using a training set with the same skewness as the test set and 50% is the balanced training set that we used before. The overall best performance is achieved by the 70% set, which has a F_1 score of 0.38 with the decision boundary of $t = -2$. If we, however, care more about recall (which we might in the 'extract the negative references' scenario) then the best we can do is the 99.9% training set with $t = -3$, which retrieves 76% of the positive examples but with only 0.12 precision.

4 Local Trust

In the model described above, we use the evaluation of a user by the entire CS community as a feature set for predicting negative references. An

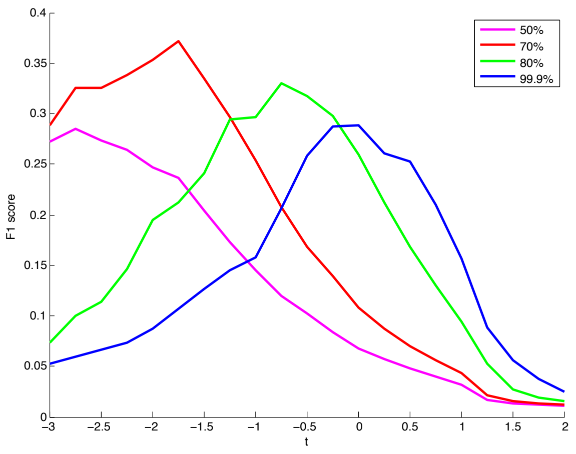
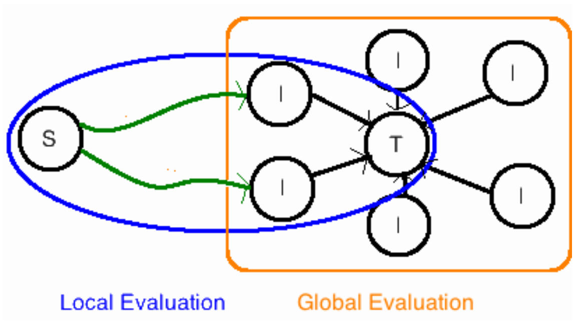


Figure 4: F_1 performance for different training sets

alternative to such a global evaluation would be to preferentially consider information from close, highly trusted friends. Maybe the wisdom of the crowds is not as good as a piece of sound advice from a trusted friend.



We hypothesize that if a user, the *source*, wants to evaluate another user, the *target*, he would get better information by considering the evaluation of the target by his own friend network than considering the global evaluation by the entire CS network. The information that comprises any evaluation stems from the set of *informants*, users evaluated the target. We model an evaluation of the target that is personalized to the source by weighing the information about the target by some function of the trust the source has in the informants. The underlying assumption is that information coming from an informant we highly trust is worth more than information from a person we don't know.

The three key components of this model are estimating the trust the source has in each informant, mapping those trust values to information weights and aggregating the weighted information from each informant to form a local evaluation.

4.1 Estimating Trust

If there is a friend connection between the source and an informant, the trust score is given explicitly in the friendship connection and their is no need to estimate trust. The CS graph, however, is very sparse; the source is friends with an informant in only 7% of the cases. To make a local evaluation richer in information, we can take another step and include friends of friends as informants. Here the problem is that we don't know how much the source trusts the friends of his friends. We only know how much the source trusts his immediate friend (t_1) and how much his immediate friend trusts the informant (t_2). In this second part of our project, we investigated how trust propagates in the CS network by designing functions that given t_1 and t_2 estimate the trust from the source to the informant (t_3). In order to fit and test our models, we extracted all 1.5M trust triads of the form (t_1, t_2, t_3) from the CS database.

The trust propagation functions are of the form $\tau(t_1, t_2) = t_2$, where $t_1, t_2 \in \{1, 2, 3, 4, 5, 6\}$ and $t_3 \in [1 : 6]$. (see Table 1 for the semantics of the trust values). We first created a simple probabilistic model. We treat trust T as a multinomial random variable. If we assume T_1, T_2, T_3 to be independent, then it follows:

$$P(T = t) = \prod_{i=1}^{6} \phi_i^{1\{t=i\}}$$

$$P(t_1, t_2, t_3) = \prod_{i=1}^{6}\prod_{j=1}^{6}\prod_{k=1}^{6} \phi_{ijk}^{1\{t_1=i, t_2=j, t_3=k\}}$$

$$P(t_1, t_2) = \prod_{i=1}^{6}\prod_{j=1}^{6} \theta_{ij}^{1\{t_1=i, t_2=j\}}$$

Using maximum likelihood estimation, we get:

$$\hat{\theta}_{ij} = \sum_{l=1}^{m} \frac{1\{t_1^{(l)} = i, t_2^{(l)} = j\}}{m}$$

$$\hat{\phi}_{ijk} = \sum_{l=1}^{m} \frac{1\{t_1^{(l)} = i, t_2^{(l)} = j, t_3^{(l)} = k\}}{m}$$

Using the definition of conditional probability:

$$P(t_3 | \, t_1, t_2) = \frac{P(t_1, t_2, t_3)}{P(t_1, t_2)}$$

$$P(t_3 | \, t_1, t_2) \approx \frac{\hat{\phi}_{t_1, t_2, t_3}}{\hat{\theta}_{t_1, t_2}}$$

Based on this model, we propose two methods for predicting t_3:

- $\tau_{avg}(t_1, t_2) \, : \, E[t_3 \mid t_1, t_2]$
- $\tau_{max}(t_1, t_2) \, : \, \arg\max_{t3} P(t_3 \mid t_1, t_2)$

We also tried three linear models:

- $\tau_{lin}(t_1, t_2) \, : \, \alpha t_1 + \beta t_2 + \gamma(t_1 \cdot t_2) + \delta$
- $\tau_{hlin\text{-}sp}(t_1, t_2)$: hierarchical linear model with (t_1, t_2) as random effect
- $\tau_{hlin\text{-}u}(t_1, t_2)$: hierarchical linear model with source user as random effect

All of our functional models except for τ_{max} rely on a sensible ordering and scaling of the input variables. One issue with the way CS encodes trust is that the value 1 corresponds to "I don't know this person well enough to decide", which is less than 2 which corresponds to "I don't trust this person". A value of 1 denotes a lack information about trust rather than a lack of trust itself. In our models, instead of treating 1 as strong distrust we treat 1 as signaling an unknown trust value, which we estimate as the average trust in the whole system (map $1 : E[T] = 4.28$).

In order to establish a performance baseline for each model, we generated a random data set with the same distribution of trust scores as in the real data set (as done in (Leskovec et al., 2010)). This allows us to disentangle the influence of the distribution of trust scores from the structure of trust triads on model performance. The results are displayed in Table 3. We provide the mean squared error for each model (MSE) as well as the difference between the MSE on the shuffled data set (δ). Overall, it appears that the structure in the trust triads does allow a better prediction. Mapping the 1 values to $E[T]$ improves performance and τ_{avg} performs best.

Trust-Function	No Mapping MSE (δ)	$1 : E[T]$ MSE (δ)
τ_{avg}	1.196 (+0.37)	0.657 (+0.16)
τ_{max}	1.611 (+0.02)	0.911 (+0.14)
τ_{lin}	1.210 (+0.36)	0.669 (+0.15)
$\tau_{hlin-sp}$	1.196 (+0.37)	0.658 (+0.16)
τ_{hlin-u}	1.460 (+0.10)	0.836 (-0.02)

Table 3: Performance of different trust propagation functions

Another concern, especially for the linear models, is that the difference in trust from one CS trust level to another may not be constant. In the attempt to find an ideal scaling we tried to design an invertible function, that would allow us to rescale the CS trust values before learning our trust propagation functions and then map the outputs of the trust propagation function back into CS trust values in order to test the accuracy of our model and compare it to the baseline CS scaling.

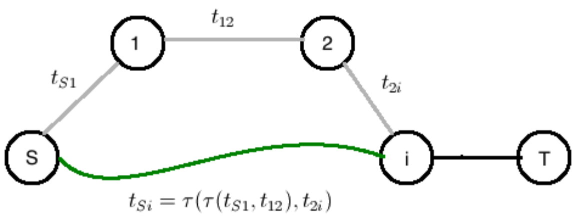
As our rescaling function, we tried using the logit function and piecewise quadratic function because they are invertible and have the ability to make high trust levels even higher while making low trust values even lower. To learn the parameters of our mapping function f we attempted to minimize the following objective using gradient descent:

$$\sum_{i=1}^{m}(f^{-1}(\tau(f(t_1), f(t_2)) - t_3)^2$$

This rescaling endeavor was rather unsuccessful as MSE under rescaling was never lower.

5 Implementing and Testing Local Evaluation

As described earlier, we can think of a trust propagation function as completing a trust triad, which is missing a single link. Trust propagation can then be applied n times to estimate the trust that S has in an informant who is $n + 1$ steps away:



This would in theory allow us to get a fully local evaluation. In practice, we only consider informants that are up to three steps away from the source. We make this "3^{rd} order" approximation due to the computational difficulty of computing every path from the source to every informant.

After computing the trust the source has in each informant it is necessary to first weigh and then aggregate the information from each informant to arrive at a local evaluation. As information weights, we tried $w_1(t) = \frac{t-1}{5}$ and $w_2(t) = 0.29 \cdot \log \frac{t-1.9}{6.1-t} + 0.5$, which both increase in trust. w_2 weighs high trust higher and low trust lower than w_1. Information is aggregated, by taking the weighted average of the information over each path from the source to an informant.

To evaluate these local trust methods, we extended our training examples with a "local evaluation of T" feature set. The performance of the four methods, as well as a global trust baseline, are displayed in Table 4. Unfortunately, we had to train on a relatively small training set ($m = 1800$) due time constraints and the computational cost of finding all local informants.

Case	w/o local evaluation	w_1	w_2
Acc	77.24	80.60	81.00

Table 4: Accuracy when training on a balanced training set $m = 1800$ and a balanced test set

For both weighting schemes, adding local evaluation features increases accuracy by approximately 4%. A problem is that only a third of the cases have at least one informant within three steps. In a real prediction system, however, one could simply use the model trained with local evaluation features if they are present and otherwise use the model trained without local evaluation features. We were unable to test the merit of local trust with more robust amounts of training data.

References

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Lauterbach, D., Truong, H., Shah, T., and Adamic, L. (2009). Surfing a web of trust: Reputation and reciprocity on couchsurfing. com. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 4, pages 346–353. IEEE.

Leskovec, J., Huttenlocher, D., and Kleinberg, J. (2010). Signed networks in social media. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 1361–1370. ACM.