SuperFan: Finding Unique Users Across Social Networks

Victor Cheung
Department of Computer Science
Stanford University
Stanford, CA
Email: hoche@stanford.edu

John Yang-Sammataro
Department of Computer Science
Stanford University
Stanford, CA
Email: johnys@stanford.edu

***Abstract*—Disambiguating users across diverse social media platforms has high potential commercial value. Our project SuperFan, is a system that can ingest a set social media profiles to produce a set of unique audience profiles by matching the profiles across social networks. We learn models that can identify unique users across diverse social media platforms. To build these models, we collect only information that's common and widely accessible given a context of increasing hostility to scraping and concerns about data security and data ownership on the part of large social networks. We overcome difficulties in generating labeled data sets by building a scraper-crawler to ingest data from social media aggregators (About.Me and Klout) and bootstrap data collection from data gathered by following additional identities that may be scraped. To make predictions, we leverage a number of string comparison metrics in order to compare similarities between user profiles. Models trained include a binary classifier under square loss, a logistic regression based classifier, a stochastic gradient boosted tree, a random forest, and a two-layer neural network. Boosted tree and random forest perform the best, with high F1 scores for matches and non-matches.**

***Keywords*—machine learning, entity resolution, social networks, disambiguation.**

I. Introduction

Social media followings are now an integral customer communication and marketing channel for brands, entertainment stars, and a new breed of Internet famous celebrities. Billions are spent each year on social networks. However, many of these Internet famous entities have little insight across their sprawling, fragmented social networks (Facebook, Twitter, Instagram, Youtube, etc.) into who their followers are, how they overlap, and more importantly how they can direct their messaging on a per a user basis across their channels to effectively monetize.

Producing a set of unique set of audience profiles, insight, and recommendations in order to interact with them is a holy grail of social media marketing in the digital age. The authors have confirmed that tens of millions to billions are being left on the table though several industry connections with whom we plan to run trails.

A. Task Definition

Our focused task is to ingest a pair of social media profiles (p1, p2) and predict whether the two profiles are the same user.

B. Literature Review

Related to this problem is that of finding duplicate records within a database, which is well-studied. These two problem domains tie together in that many of the techniques for identifying duplicate entries and duplicate users are shared — recent work tackling duplicate users across social networks have also built on the database literature [1]. Proposed applications in the literature include attacks to de-annoynimize users, finding user profiles on different social networks given an example profile, and labeling pairs of profiles from different networks.

Broadly, there are three kinds of features that have been considered: 1) social-based, group membership and graph topology; 2) name-based, leveraging edit-distances; 3) information-based, such as distances between physical locations, employers, autobiographies, etc. Notably, Peled et al. [1] leverages all three types of features to compare users across Facebook and Xing. The most surprising finding therein was that the top five features contributing to AUROC (Area Under Receiver Operating Curve) were all based on names. This is in spite of the large amount of data gathered on other characteristics, such as list of friends and friends of friends, which is exponential in memory usage. Peled et al. also provides an excellent and exacting definition of social networks.

Other work, namely Wondracek et al. [2], have focused on teasing out user identity via group membership information alone. It assumes that an attacker has access to a set of "groups" visited by some user that slowly grows overtime as the attacker observes browsing behavior. The user is assumed to be a member of at least one of those "groups". And assuming we have group membership information (i.e. the set of users in some group), we can then infer a common set of users who have visited the entire set of groups. When that common set shrinks to size one, we have de-annoymized the user. Practically speaking, the attack vector used and crawlers designed by Wondracek et al. may no longer be feasible in 2016, as social networks have doubled down on protecting network data. There is no reliable way to crawl the entire membership directory of groups in order to set up the database required for the attack. Furthermore, maintaining an up-to-date database presents may be another serious challenge.

An approach with more conservative data requirements was presented in 2012 by Malhotra et al. [3], which leverages userid, name, autobiographies, location, connections and pro-

file pictures for what they called user-profile disambiguation. Similar results to [1] were found in that name-based features contributed most to discrimination. Their data sources were primarily Twitter and Linkedin, both of which remained attractive for scraping at the time. The latter has since then been acquired by Microsoft, and long before then had begun implementing serious anti-scraping measures.

Finally, in [4], Raad et al. undertook an experimental approach and generated data simulating real social networks. This allowed greater flexibility and control in studying how differences between profiles drove differences in decision-making in classifiers; it should not be surprising, however, that the underlying takeaways were largely the same as the prior studies. That is, profiles that were similar were highly similar (in name, user-names, location, autobiographies, etc.) and profiles that were dissimilar were highly dissimilar.

In general, researchers have had to grapple with fewer data points with less information each as time went on. This trend coincides with a parallel trend of growing concerns with data security and data monetization, as social networks began to realize the value of the goldmines on which they sat. Our contribution to the literature is in an approach that maximizes discriminative ability between users all the while using only the smallest subset of information that is common between most social networks.

C. Overview

We will discuss in section II our infrastructure, which aimed to collect common data across a diverse set of social networks, and its associated challenges. In section III we take a brief tour of the shape and form of the data, the features to be implemented, and the pre-processing necessary to fit certain assumptions for models later used. In section IV, we briefly describe our approach in learning the best models for predicting matches between user profiles. In section V, we examine the empirical performance of our models on a test set and analyze the type of and reason for errors seen. In section VI, we will conclude with next steps and future extensions of this work.

II. Infrastructure

As our approach will be through supervised learning, our infrastructure is designed around one core theme: how to obtain labeled data? For example, Facebook boasts billions of users, and Twitter hundreds of millions. If we blindly scraped even hundreds of thousands of profiles from the two, the chances that we happen upon a duplicate profile are worse than one in a trillion. The APIs that past papers have relied on to circumvent this problem, such as Google's Social Graph API, are no longer available. We needed a central directory of users who also provided links to other profiles they owned. Fortunately, the existence of sites like About.me, which is intended to be an aggregator of online identities, made data collection possible.

Thus, one of the most challenging and exciting parts of the project has been the development of an extensive infrastructure pipeline according to the following ETL process.

1) Extraction - Profiles and Match Labels: We developed a scalable scraper to (1) scrape profiles and profile matches

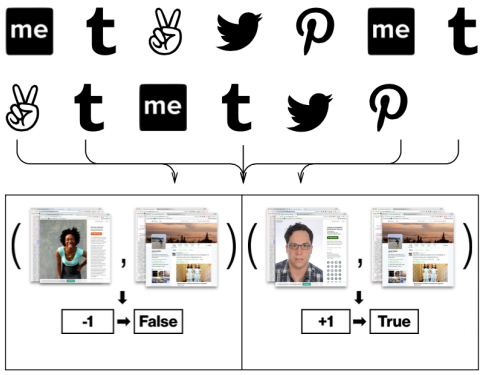

Fig. 1. Data ingestion and classification

```
featuresDB =
{
  udid:
  {
    OSN_'username'
    First_name:
    last_name:
    location:
    Bio:
    networks:
    {SN1: 'username',
    SN2: 'username'
    }
  ...
}
```

Fig. 2. Features Database

to other channels from About.me, (2) scrape profiles from Twitter (3) scrape profiles from other social networks linked on About.me (Youtube, etc.) The program then saves all the raw extracted information in JSON for the transformation phase. In particular, this extraction phase is self-sustaining for a given user as long as new networks as observed — for example, once we scraped the twitter link from an About.me profile, we follow the twitter link and scrape the Pinterest link from the twitter profile, and so on.

2) Transformation - Data Cleaning : Our transformation phase cleans and imposes standard structure on raw JSON data. It uses extensive data wrangling and error catching to convert raw JSON into a structured python dictionary of all the user profiles. This process included several steps and additional scraping to retrieve profile information outside of About.me (e.g. Twitter) and outputs a serialized version of the following featuresDB dictionary:

3) Load - The final step in our infrastructure pipeline takes the serialized cleaned and structured data above and creates pairsDB and vectorsDB (outlined in the Approaches section) from the information to load it into our models.

A. Challenges

Social networks are increasingly reluctant to make data available publicly, even for academic purposes. Hence, data-scraping presented a significant challenge.

1) About.me Scrape: We reverse engineered part of About.mes Alogia search API, raw scraped a dynamically generated part of the site, and integrated with Twitters API. This process took several days to develop since there are few sites which have labeled social media profiles. Others have anti-scraping setups, ex. Facebook or are behind a login with strict rules. Our initial set of labeled data therefore consists of matching About.me to Twitter profiles. Our scalable foundation will allow us to expand this to any other social networks with similar characteristics to About.Me.

2) Python Object Caching and Serialization: Another aspect of our infrastructure pipeline is that we have developed a set of processing modules that each take in network and/or python serialized pickle objects and output python serialized pickles that can be stored on disk. Our current 3 step ETL pipeline actually has 5 intermediate steps (raw about.me directory data, about.me structuring, about.me profile retrieval, twitter profile retrieval, data transformation and loading for models). 3 of the 5 steps include significant networking calls prone to blocks or failure and all 5 steps have lengthy execution. The process of loading input and pickling at each step, allows us to save intermediate results. This has the benefits of a flexible and modular development pipeline, iterating on intermediate results or swapping out pipeline modules which has streamlined our progress and allows us to scale our data ETL infrastructure for the final stage of the project.

III. Data Description and Preprocessing

A. Data

Our final dataset consists of 112,600 users across six social networks: AngelList, Twitter, Klout, About.me, Facebook and Pinterest. The set of information we judged readily available on all publicly-facing social networks are as follows:

1) Usernames: the unique handle for users on social networks. For example, "johnys" is the twitter handle of one of the authors, and "victoration1" is the handle of the other. This information is always exposed over the web as a unique key into the profile of the user. Furthermore, collisions between usernames are disallowed within most social networks (hence the error message "Sorry, this username is taken."). This bodes well for our classification task, as one of our assumptions is that there are no duplicate profiles within the same social network.

2) Full names: the full name of users. For example, "Snoop Dogg" is the full name scrapped for Snoop Dogg on Twitter. Notably, there are no restrictions against name collisions for legal names, so that two profiles with "John Smith" as the full name may very well be different people.

3) Location: the geographic location given. For example, "Washington, D.C." is scraped when examining the profile of "POTUS" on Twitter. Past work in the literature have constructed altitude and longitude coordinations for locations, but we believe that locations are often ambiguous enough to make any single guess incorrect (does "Vancouver" refer to "Vancouver, British Columbia" or "Vancouver, Washington"?). They also depend on the continued availability of services like Google Map's API. In keeping with our minimalist approach, we take the location given at face value, i.e. as a string that may or may not contribute to disambiguation.

4) Autobiographies: more informally as "bios" or "about me". These are short snippets often available on a profile that are user-generated content. Insofar as the assumption holds that users provide similar biographies consisting of largely the same words across different social networks, then this may contribute substantively to user disambiguation as a unique fingerprint beyond usernames and full names.

5) Networks: this is a list of networks that a user may have provided. This information enables us to bootstrap the data collection process.

B. Features

In the context of classification across comparisons of users, weights assigned to specific characters or strings are meaningless. We require a way to compare the difference between strings. Fortunately, the literature in entity resolution as well as database deduplication provides a fertile starting ground. Much work has been done in information theory to devise measures of similarity between strings. We exploit a subset of those that apply well to the task at hand. As a note, given the relative small size of our dataset, the use of naive feature templates such as indicator functions for comparison across specific characters, n-grams and so on may result in overfitting; this is due to the blow up in the number of naive features to be learnt. The string similarity measures used below are numeric representations of the same underlying comparison.

Variations in usernames are likely best captured by edit distances like Levenshtein and Damerau-Levenshtein, which explicitly consider the number of the edits needed to transform one string to another. Mispellings are likely best represented by Jaro and Jaro-Winkler metrics, which take a different philosophy and captures more of a proximity between strings in terms of positions of characters and transpositions.

1) Levenshtein Distance is the number of edits (consisting of substitutions, deletions and insertions) necessary to change some string a to another string b. Thus, this distance will always return a whole number upper bounded by the difference in length between the two strings. Formally, it is defined as

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & min(i,j) = 0 \\ \min(x,y,z) & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} x &= lev_{a,b}(i-1,j)+1 \\ y &= lev_{a,b}(i,j-1)+1 \\ z &= lev_{a,b}(i-1,j-1)+[a_i \neq b_j] \end{aligned}$$

2) The Damerau-Levenshtein Distance is the number of edits (consisting of substitutions deletions, insertions and transpositions). As evidenced by the name, Damerau expanded on the Levenshtein distance by allowing transpositions as a valid edit operation. Damerau originally intended the modification to capture the most common reasons for misspellings. We choose this distance measure for precisely that reason. Users may misspell names, locations, etc. between social networks. Effectively, we capture differences in syntax (with the same semantics) with this distance measure. This measure will return

a whole number also upper bounded by the difference in length between the two strings.

$$dl_{a,b}(i,j) = \begin{cases} \max(i,j) & min(i,j) = 0 \\ \min(x,y,z,w) & \text{if } i,j > 1, a_{i-1} = b_j, b_{j-1} = a_i \\ \min(x,y,z) & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} x &= dl_{a,b}(i-1,j)+1 \\ y &= dl_{a,b}(i,j-1)+1 \\ z &= dl_{a,b}(i-1,j-1)+[a_i \neq b_j] \\ w &= dl_{a,b}(i-2,j-2)+1 \end{aligned}$$

3) The Jaro Distance is a string similarity measure that captures the proximity between characters in two different strings. It makes an allowance for transpositions by explicitly discounting them. It's defined formally as follows

$$jaro(s_1,s_2) = \begin{cases} 0 & m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

Matching characters m is defined as two characters that are the same and within $\lfloor \frac{\max(|s_1|,|s_2|)}{2} \rfloor - 1$ characters of each other. Transpositions t is defined by the number of matching characters that are in different order, divided by two. So Jaro Distance is not an exact measure of the number of edits required to change one string to another - it instead captures the "proximity" between two strings via the proximity of its character components and the number of characters in the wrong order. For this reason, we anticipate that subtleties in name variation will be better captured by Jaro distance than for pure edit distances alone.

4) W.E. Winkler, modified the Jaro distance above based on the observation that strings which match at the beginning are more likely to be the same - with the corollary that most misspellings or variations tend to occur towards the middle or the end of strings. In effect, he placed an additional weight on the prefix—of some length—of strings in the case where they match. This is defined as

$$jw(s_1,s_2) = jaro(s_1,s_2) + lp(1 - jw(s_1,s_2))$$

where l is the length of the common prefix not exceeding a length four, and p is the additional weight provided for having a common prefix, not to exceed 0.25. Note $p = 0$ is a reduction to Jaro distance.

5) The Hamming Distance between two equal length strings is the number of substitutions required to change one to the other; in other words, it's the number of positions where the two strings do not agree. Formally, for two strings a, b of length n

$$Hamming(a,b) = \sum_{i=1}^{n} [a_i \neq b_i]$$

6) Jaccard similarity is a general measure of the similarity between elements of a set. In this case, where we examine biographies and location of users, simple edit distance may not suffice as these are ill-adapted to longer string comparisons. We therefore look at the similarity of words used throughout biographies. Jaccard similarity is formally defined, for two sets A, B as

$$Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

So it's clear that the similarity is trapped between zero and one, where one indicates that the two sets have precisely the same elements, and zero indicates they have nothing in common. We note that Jaccard similarity is order-invariant.

C. Preprocessing

We construct a labeled dataset called PairsDB as follows. For each user profile x in featuresDB, if the networks field isn't empty and we can find a corresponding profile y on a different network for the same user that's also in featuresDB, then we add a new pair $(x, y) : 1$ to PairsDB. For each pair added in this way, we randomly sample from featuresDB to add two more pairs $(w, z) : 0$ and $(w', z') : 0$ to PairsDB that we know are not matching users. This is in effect a stratified sampling procedure that allows us to oversample matches and under-sample non-matches. This also changes the prior distribution of matches and non-matches and will affect the relative costs of false negatives to false positives later on.

We construct similarity vectors as follows. For each pair $p = (x, y)$, let A_p denote the set of Jaro distance metrics computed on username, first name, last name and location; let B_p denote the set of Levenshtein distance metrics computed on the same; let C_p denote the set of Damerau-Levenshtein distances computed on the same; let D_p denote the set of Hamming distances computed on the same; let E_p denote the set of Jaro-Winkler distances computed on the same; and finally, let F_p be the set of Jaccard distances computed on the autobiographies. Then each similarity vector is $v_p = [A_p, B_p, C_p, D_p, E_p, F_p]$. And VectorsDB is precisely the similarity vectors for all pairs in PairsDB.

To construct our training data, we randomly sampled without replacement 80% of PairsDB. The other 20% is reserved as a test set to estimate the generalization error.

We note that similarity measures differ in their magnitudes. For example, without scaling, Demerau-Levenshtein distance and Levenshetein distance are whole numbers; in contrast, Jaro distances are trapped between 0 and 1. This implies that feature vectors are "pulled" along certain dimensions by implicitly placing greater emphasis on features with larger range. In particular, this interferes with the derivation of optimal weights for regularized models. Predictions of models like Neural Networks and Logistic Regression—in fact, any model with possible regularization toggled as a hyperparameter—would be affected thus. To work around absolute differences in range, we apply a transform to zero mean and unit variance trained on the training data. We apply the same transform to the test data.

IV. Approach

We assume there is some true $f(X)$ that generates the correct label [5]. This is familiar from regression theory. We do not know either the functional form of f nor the full set of variables/predictors X that is its input. This f can

be interpreted as the true response function that we seek to estimate in various ways, or equivalently, as the perfect estimator with zero generalization error.

We attempt to find a predictive model, or what Berk [5] calls a forecasting model, based on the data that we have collected. The closer that our estimator is to the true $f(X)$, the better our performance both in sample and out of sample. However, we know that for a complex enough f and a small dataset, that from learning theory it's highly unlikely we manage to find f; the best we can do is to inform a decent approximation of it.

Given a multitude of widely used, highly adaptable, open-source machine learning libraries, we use Scikit-Learn [6] to construct our models, conduct cross-validation, and Matplotlib for graphics generated.

The following section explains each of the four approaches to modeling we explored. All results are in section V.

A. Baseline and Oracle

Our baseline—the most naive yet workable strategy–is a straightforward username comparison. People may have the same first names and last names, and in social networks the probability of two people having the same legal name approaches unity as the network size increases to infinity. Then this is a poor method for an injective unique user matching. Fortunately, social networks often enforce the policy that usernames be distinct. Users tend to adopt the same usernames across networks. From experimentation, this naive approach results very high precision - positives. However, we cannot pick up on even slight differences in usernames, even when prima facie it's immediately clear two usernames are highly similar.

Our oracle is simply a human examination of both profiles. This approach leverages information we do not have for the models, such as profile pictures, comment histories, and so on. As well, experimentation shows that humans pick up on information otherwise not exploited. For example, one user's comment history may be indicative of a highly cheerful personality. The other may be more somber, and consist largely of current events. This provides evidence against the hypothesis that these two accounts belong to the same user — provided one considers that one may be a professional network. Our oracle thus has close to a 100% recall and precision when shown two profiles.

B. Generalized Linear Models

Under GLMs we use linear regression and logistic regression models. Both are in the exponential family.

Linear regression allows us to construct a binary classifier under square loss. That is, we minimize the loss

$$Loss_{square}(w) = \sum_{i}^{n} (w \cdot \phi(X_i) - y_i)^2$$

And for each example, $sign(w \cdot \phi(X_i))$ is the predicted label [7]

Logistic regression is more natural for binary classification — in this instance, we have an objective function minimizing the logistic loss that's L2-regularized

$$Loss_{Log}(w, C) = \frac{1}{2} w \cdot w + C \sum_{i=1}^{n} \log(\exp(-y_i(w \cdot \phi(X_i) + c)) + 1)$$

L2 regularization implies that we prefer to flesh out weights for features, but keep the weights small. Sparse models under L1 would make not make sense here, as we have many features that are just transformations on the same underlying data. For each example X_i, the label assigned is $+1$ if $P(y = 1; w) = \sigma(w \cdot \phi(X_i)) > 0.5$ and -1 otherwise. Note that σ is the familiar logistic sigmoid function [7]

C. Non-Parametric Approaches

By nonparametric we mean that we assume that the data does not derive from some known and well-defined distribution. Instead, we allow the data to dictate our prediction. In this case we use CART (Classification and Regression Trees) as first devised by Breiman in 1986 [5] and Neural Networks. Both models impose nonlinearity between the input and the output: neural networks via activations and CART via its adaptive nearest neighbor fitting [7] [5].

Breiman intended CART to be interpretable—after recursively partitioning to obtain the best splits (to maximize information gain or to minimize impurity within nodes), the result is a kind of tree down which one can follow the paths to determine a classification for an example. The appendix contains the CART built on the training dataset.

Neural networks are complex learning algorithms that learn hidden layers of transformations and representations before a final output. The model we trained has two hidden layers, the first with twenty nodes and the second with ten. Our activation function is the rectified linear unit function $max(0, x)$, $x \in \mathbb{R}$. We imposed a large L2 regularization penalty at $\alpha = 1$ (the default $\alpha = 0.0001$) to avoid over-fitting. The objective function to be minimized is the L2-regularized log-loss, similar to the logistic regression above.

D. Ensemble Methods

Ensemble methods have achieved quite successful results in prediction and popularity in usage despite being black box methods, much like neural networks. We use random forests as first devised by Breiman and stochastic gradient tree boosting (SGBT) as originally devised by Friedman.

Random forests build a forest of CARTs to maximum depth (to minimize bias) with variance controlled by having classification decisions made democratically via majority voting. In particular, each CART is trained on a "bagged" (bagging creates multiple new samples by sampling with replacement from some original dataset) sample of the training data, with the rest of the data not used (referred to as Out-of-bag, or OOB) used for scoring and for assignments with respect to that particular CART. Furthermore, for each split in a given CART, a random sample of the features (predictors) is taken without replacement to be considered as a candidate for the split. These procedures theoretically lower both bias and variance

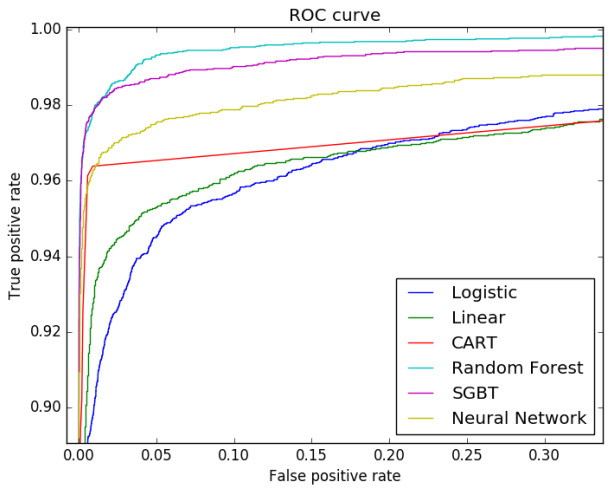

Fig. 4. Receiver Operating Characteristics

TABLE I. PRECISION - RECALL TABLE ON TEST DATA

	-1			+1		
	Precision	Recall	F1	Precision	Recall	F1
Baseline	1.00	1.00	1.00	1.00	0.26	0.42
Oracle	1.00	1.00	1.00	1.00	1.00	1.00
Neural Network	0.98	1.00	0.99	0.99	0.95	0.97
Boosted Tree	0.98	1.00	0.99	0.99	0.97	0.98
Random Forest	0.98	1.00	0.99	0.99	0.97	0.98
Logistic Reg.	0.97	0.99	0.98	0.99	0.93	0.96
Linear Reg.	0.94	1.00	0.97	1.00	0.87	0.93

for random forests [5], and renders them some of the best classifiers that can be constructed.

We used boosting as an ensemble algorithm that can convert weak learners, classifiers that do only slightly better than random, into strong learners, classifiers that can be theoretically trained to arbitrary accuracy. In our case, we use decision stumps, or trees with minimum depth, as our weak learners. Boosting is in some ways similar to random forests—both focus on local features of the data that may not easily be captured by a single classifier. In our case, we applied stochastic gradient boosting, which in effect means that we trained many many iterations of decision stumps on bootstrapped samples of the data, each time focusing on instances that were misclassified. The final output is a weighted combination of stumps from all iterations, with stumps that were more correct given higher weights [5].

To determine the appropriate hyperparameters for both boosted trees and random forests, we conduct a randomized 5-fold cross-validated search over a subset of the hyperparameter space specified by hand. The results are embedded in the models.

V. RESULTS AND ERROR ANALYSIS

We give brief overviews of the results here. First, some definitions. Let tp be the number of true positives, fp be the number of false positives, tn be the number of true negatives and fn the number of false negatives for a given classifier. Then

TABLE II. CONFUSION TABLE ON TEST DATA

Truth	-1		+1		Relative Cost
Prediction	-1	+1	-1	+1	
Neural Network	11263	32	280	5414	8.75
Decision Tree	11238	57	254	5440	4.45
Boosted Tree	11246	49	180	5514	3.67
Random Forest	11258	37	187	5507	5.05
Logistic Reg.	11229	66	403	5291	6.11
Linear Reg.	11276	19	726	4967	38.2

1) Precision p is the ability of a model to discriminate between the two labels; it's the fraction of matches found that are actually correct. Hence,

$$p = \frac{tp}{tp + fp}$$

2) Recall r is the ability of a model to identify true positives; it's the fraction of all true positives that a model manages to find. Hence,

$$r = \frac{tp}{tp + fn}$$

3) F1 score is a combination of the precision and recall scores; in fact, it's the harmonic mean between the two. Hence

$$F1 = \frac{2pr}{p + r}$$

4) Relative cost rc is the ratio of false negatives to false positives. It's indicative of which types of mistakes are more costly - eg. a relative cost ratio of ten means that false negatives are ten times cheaper than false positives, or that false positives that ten times more expensive than false negatives. Consequently, our model is, in some sense, more willing to classify a given example as a non-match than as a match. Hence, relative cost

$$rc = \frac{fn}{fp}$$

From examining table I, we see that our models are excellent at discriminating between matches (+1) and non-matches (-1). In fact, in our stratified test set, we see that even our worse models are able to distinguish non-matches the vast majority of the time. As expected, the baseline model was highly precise but had poor recall rate — this is due to the exacting evidence required for the baseline algorithm to call a match. As we lessen the burden of evidence, we aim to achieve high recall and high precision. A priori it's not clear at all that we can raise both recall and precision. The two are typically inversely related. In this particular instance, we find that boosted trees and random forests represent the best compromise between precision and recall — both have the best F1 scores across the matches and non-matches. This is another piece of evidence suggesting that, at least in classification tasks, it's often difficult to best random forests and boosted trees [5].

Another way to assess the discriminative power of our models is through the Receiver Operating Characteristics curve, as in figure 4, careful scrutiny will show that even between the worst performing models (linear regression) and the best performing models (stochastic gradient boosted tree or random forest) the area under ROC differ very little. That is, given two pairs, one of which we know is a match, and the

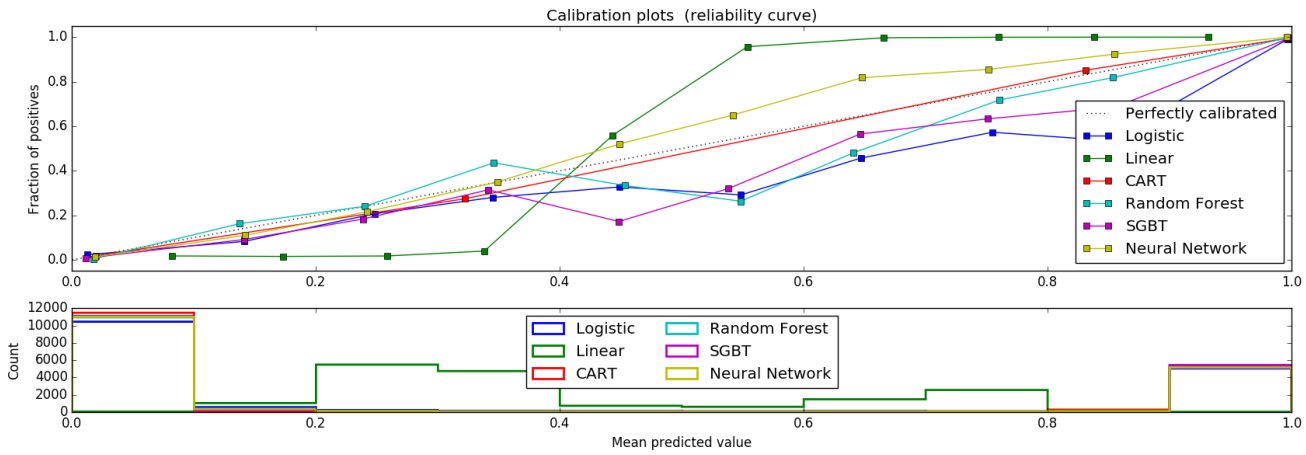

Fig. 3. Calibration Curve

TABLE III. FALSE POSITIVE EXAMPLE UNDER SGBT

	Username	First name	Last name	Location	Bio
User A	afelina777	Albina	Katrick	United States	I am cheerful, cheerful person, and I love to travel and cook.
User B	baylytaylor	Bayly	Taylor	N/A	I love my family, friends and God =)

TABLE IV. FALSE NEGATIVE EXAMPLE UNDER SGBT

	Username	First name	Last name	Location	Bio
User C	carrycooker	Carry	Cooker	OHIO	I LOVE COOKING! HOW ABOUT YOU?
User D	rowbvp	Ryan	Williams	N/A	I always follow back. a RN and a video game developer. Author of Zombie Slayer Books

other we know isn't a match, our models are able to pick out the match the vast majority of the time [8].

A good classifier ought to stem from good probability estimates [9]. One way to assess the underlying probability estimates is through the calibration curve, as in figure 3. For example, amongst the pairs in our dataset that a classifier considers having 75% probability of being a match (the x-axis), if 75% of those pairs are actually matches (the y-axis), then our classifier is considered well-calibrated. Consider figure 3 then. The diagonal line across the center is the reference line; curves dipping below it are overestimating the percentage of matches, and curves rising above it are underestimating the percentage of matches. For this reason, classifiers producing sigmoidal (S-shaped) curves are considered under-confident, while those producing inverse sigmoidal (sigmoidal flipped along the diagonal) curves are consider overconfident.

From figure 3, we see that classifiers based on linear regression and neural networks were under-confident; those based on boosted trees, random forests and logistic regression were overconfident. CART turns out to be very well-calibrated along the diagonal. In either case, we see that all models excepting linear regression classify most examples as either highly unlikely to be a match or highly likely to be a match. There is no middle ground. This matches our intuition. Given names, location, and bios, in most cases we can instantly say whether two users are the same or not.

To concretely see where our models may be led astray, consider tables III and IV respectively showing an instance of a false positive and of a false negative. These misclassification come from the SGBT model.

Consider Table III. In this particular case, we see similar enough name based features. Prima facie, while the names are clearly not the same, they're similar enough in sharing common letters in the correct positions or right orders that our string similarity metrics would have been relatively high. Furthermore, the autobiographies of both users, while short, share three words in common: "I, and, love" and convey largely the same kind of optimism.

Consider table IV. By construction we know that this pair must have been a match, i.e. that both profiles belong to the same person. From examination, however, there is scarce evidence to indicate this is the case. Name based features are different across the board, and the autobiographies are entirely unrelated. This is illustrative of the limitations of our data collection approach — the five pieces of data we collect may be common and easy to obtain from most social networks, but there are edge cases where this may not actually provide enough information for us to recognize that users are the same.

Given the effects of stratified sampling, our training sets and testing sets all have the same artificial 1:2 ratio as discussed in prior sections. That is, our dataset was constructed in triples, where each triple includes a matched pair and two non-match pairs. This is in effect an imposed prior over the distribution of labels over the data. Imposing this prior will have affected the relative costs observed in the confusion tables (note that relative costs and priors are not the same). Indeed, in a real world setting, if we had treated the cost of false negatives and false positives the same, then it would be difficult

to outperform a naive majority algorithm. Choosing profiles uniformly at random across large social networks like Twitter and Facebook will result in a match on the order of once per trillion pairs chosen.

As it stands, the relative cost ratio observed ranges from 3.67 to 38.2. In general, our models are more likely to classify a given pair as a non-match than as a match, though the costs incurred in making an incorrect decision varies significantly. Determining an exact cost ratio we'd like applied will depend on the specific use case for our models. Intuitively, it seems sensible that a false positive here should be so much more heavily penalized than false negatives — the threshold of evidence we should require for calling a match is substantially higher than that required to call a non-match, given the natural priors of matches and non-matches. For example, a use case that may necessitate higher relative costs is personalized messages and targeted advertisements. When sending complementary messages to different social media profiles of the same person, the damages from leaking sensitive information is high.

In terms of practical usage, for instance in implementing a web API that would allow one to calculate in batches labels for pairs of users, a relative cost ratio of 5:1 seems sensible. Choosing from our models then, the stochastic gradient boosted tree has approximately the correct cost ratio and excellent F1 scores on both non-matches and matches. It's AUROC and discriminative ability between matches and non-matches fall just below random forests as in figure 4. A potential caveat, however, is that it's an overconfident classifier, as in figure 3.

Finally, figure 5 through 7 demonstrate some characteristics of our tree based models.

VI. Conclusion

We have achieved moderate success with our models in user disambiguation across social networks. Based on a stratified dataset with 1:2 match to non-match ratio, our best models achieved average F1 score > 0.98 across both matches and non-matches. We find that this is due to the nature of the problem, which leaves little room for ambiguity between matches and non-matches. However, further testing on real world datasets with more stringent prior restrictions (eg. 1:99 ratio, 1:999 ratio) and assessment of recall and precision there will be necessary prior to any actual deployment of the models.

Future work may be concerned with allowing comparison across domain-specific information where they may arise. For example, we may have to discard information that could have helped in informing a classification decision in order to fit our framework. However, the information discarded (eg. links to profile pictures) seems wasteful. Extensions may include training classifiers on all the information that is available. This might include extending our data pipeline and generating further methods for feature extraction.

In the realm of application, we hope to use this work as the foundation for expanded systems related to social media media marketing to unlock potential commercial value, for instance, creating a system which allows for social media marketing profile matching in the same manner that drawbridge matches devices for marketing.

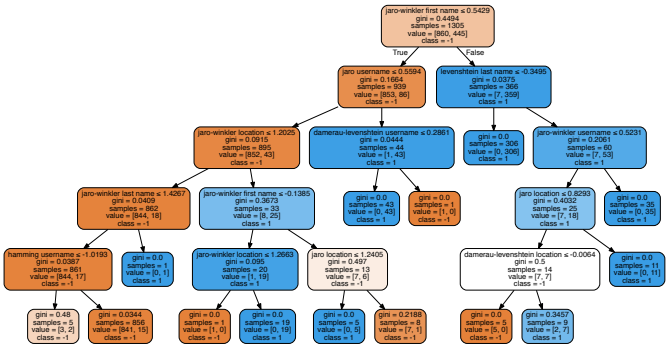

Fig. 5. CART at depth 6 showing best splits amongst features used. Note that as feature values have been scaled they may not conform to original upper and lower bounds. This applies to Jaro Winkler distances.

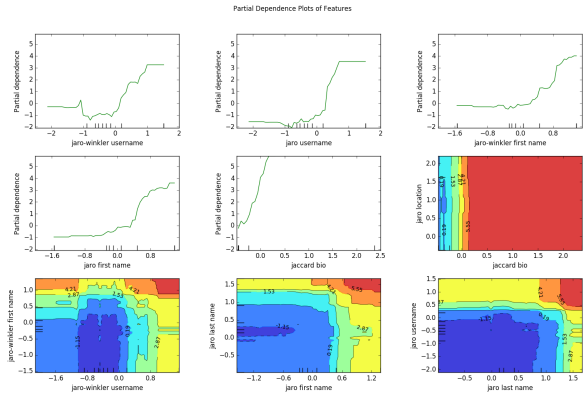

Fig. 6. Partial Dependence plots for the SGBT (boosted tree) model. This plot shows the input-output effect of certain features. We see clearly the nonlinearity of relationship between features and classification decisions. Note also the nonlinear interaction effects between features, as in the last four graphs. The prevalence of cliffs in the partial dependence plots suggests that our models automatically picked up thresholds that drive labeling decisions that minimized specific loss functions without manual hand-tuning, as desired.

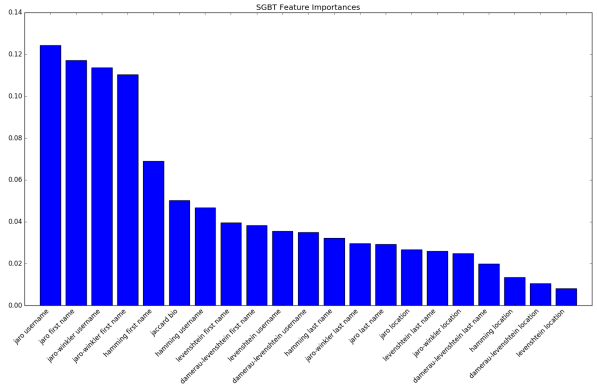

Fig. 7. Feature Importance plot for the SGBT (boosted tree) model. This plot shows the contribution of each feature in the classification decision. It allows a ranking of the features in order of importance [5]. It turns out that name based features are the most critical, and information represented through Jaro and Jaro-Winkler distances are amongst the top five.

ACKNOWLEDGMENT

The authors would like to thank Whitney LaRow for her assistance throughout this project, for fielding our many questions and for her flexibility and accommodation. We would also like to thank Prof. Percy Liang for his unwavering enthusiasm in teaching this course, for the well-prepared course materials and for his wry humor, which were often a welcome break.

REFERENCES

[1] O. Peled, M. Fire, L. Rokach, and Y. Elovici, "Matching entities across online social networks," *Neurocomput.*, vol. 210, no. C, pp. 91–106, Oct. 2016. [Online]. Available: https://doi.org/10.1016/j.neucom.2016.03.089

[2] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to de-anonymize social network users," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 223–238. [Online]. Available: http://dx.doi.org/10.1109/SP.2010.21

[3] A. Malhotra, L. Totti, W. Meira Jr., P. Kumaraguru, and V. Almeida, "Studying user footprints in different online social networks," in *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, ser. ASONAM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1065–1070. [Online]. Available: http://dx.doi.org/10.1109/ASONAM.2012.184

[4] E. Raad, R. Chbeir, and A. Dipanda, "User profile matching in social networks," in *Proceedings of the 2010 13th International Conference on Network-Based Information Systems*, ser. NBIS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 297–304. [Online]. Available: http://dx.doi.org/10.1109/NBiS.2010.35

[5] R. A. Berk, *Statistical learning from a regression perspective*. New York, NY: Springer Verlag, 2008. [Online]. Available: http://www.loc.gov/catdir/enhancements/fy0906/2008926886-d.html

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006. [Online]. Available: http://research.microsoft.com/en-us/um/people/cmbishop/prml/

[8] F. E. Harrell, Jr., *Regression Modeling Strategies*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[9] B. Zadrozny and C. Elkan, "Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 609–616. [Online]. Available: http://dl.acm.org/citation.cfm?id=645530.655658